10 SQL Tricks

That You Didn’t Think Were Possible

10 SQL Tricks

To Convince You that SQL is Awesome

Why are you here?

Because clickbait headlines work!

... Trick #3 Changed work forever!

... Trick #8 Makes doctors hate you!

... Trick #9 is bigger than life!

Me – @lukaseder

- Founder and CEO at Data Geekery
- Oracle Java Champion
- JUG.ch Board Member

> “SQL is a device whose mystery is only exceeded by its power!”

Why do I talk about SQL?

SQL is the only ever successful, mainstream, and general-purpose 4GL (Fourth-Generation Programming Language)

And it is awesome!

Why doesn't anyone else talk about SQL?

Why doesn’t anyone else talk about SQL?

¯_(シ)_/¯

What is SQL?

What is SQL?

SQL is the original microservice

What is SQL?

SQL is the original microservice

Just install a single stored procedure in an Oracle XE instance, deploy, done.

Who thinks this is SQL?

```sql
SELECT *
FROM person
WHERE id = 42
```

Who thinks this is SQL?

```
@Entity
@Table(name = "EVENTS")
public class Event {
  private Long id;
  private String title;
  private Date date;

  @Id
  @GeneratedValue(generator = "increment")
  @GenericGenerator(name = "increment", strategy = "increment")
  public Long getId() { /* … */ }

  @Temporal(TemporalType.TIMESTAMP)
  @Column(name = "EVENT_DATE")
  public Date getDate() { /* … */ }
```

Or this...?

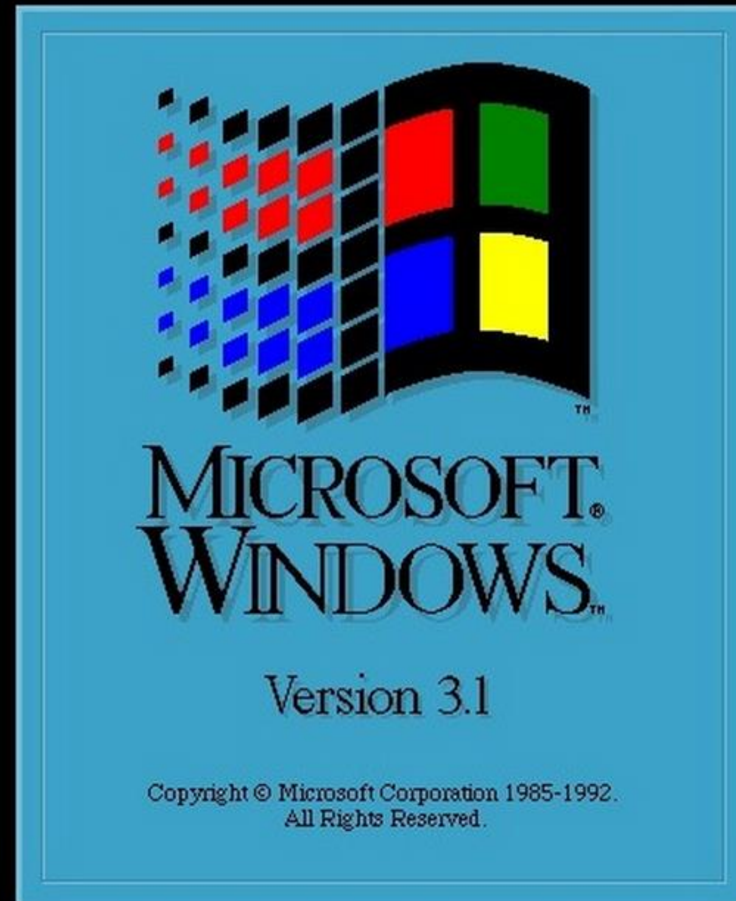
```
@OneToMany(mappedBy = "destCustomerId")
@ManyToMany
@Fetch(FetchMode.SUBSELECT)
@JoinTable(
    name = "customer_dealer_map",
    joinColumns = {
        @JoinColumn(name = "customer_id", referencedColumnName = "id")
    },
    inverseJoinColumns = {
        @JoinColumn(name = "dealer_id", referencedColumnName = "id")
    }
)
private Collection dealers;
```

Found at http://stackoverflow.com/q/17491912/521799

Think again!

Still using
Windows 3.1?

So why stick to
SQL-92?



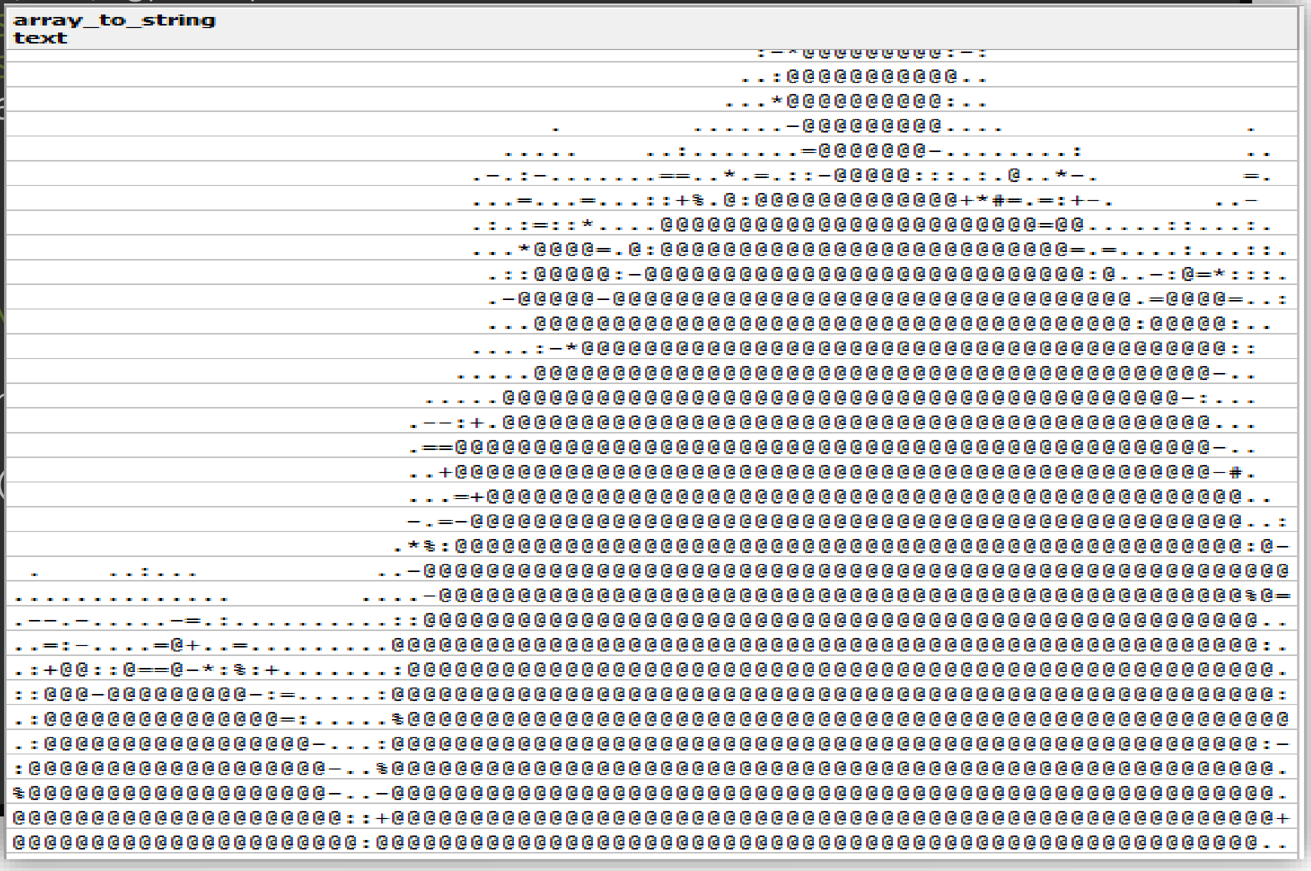
Modern SQL in PostgreSQL
@MarkusWinand

This is also SQL

```sql
-- Query from http://explainextended.com/2013/12/31/happy-new-year-5/
WITH RECURSIVE q(r, i, rx, ix, g) AS (
  SELECT r::DOUBLE PRECISION * 0.02, i::DOUBLE PRECISION * 0.02,
        .0::DOUBLE PRECISION      , .0::DOUBLE PRECISION, 0
  FROM generate_series(-60, 20) r, generate_series(-50, 50) i
  UNION ALL
  SELECT r, i, CASE WHEN abs(rx * rx + ix * ix) <= 2 THEN rx * rx - ix * ix END + r,
               CASE WHEN abs(rx * rx + ix * ix) <= 2 THEN 2 * rx * ix END + i, g + 1
  FROM q
  WHERE rx IS NOT NULL AND g < 99
)
SELECT array_to_string(array_agg(s ORDER BY r), '')
FROM (
  SELECT i, r, substring(' .:-=+*#%@', max(g) / 10 + 1, 1) s
  FROM q
  GROUP BY i, r
) q
GROUP BY i
ORDER BY i
```

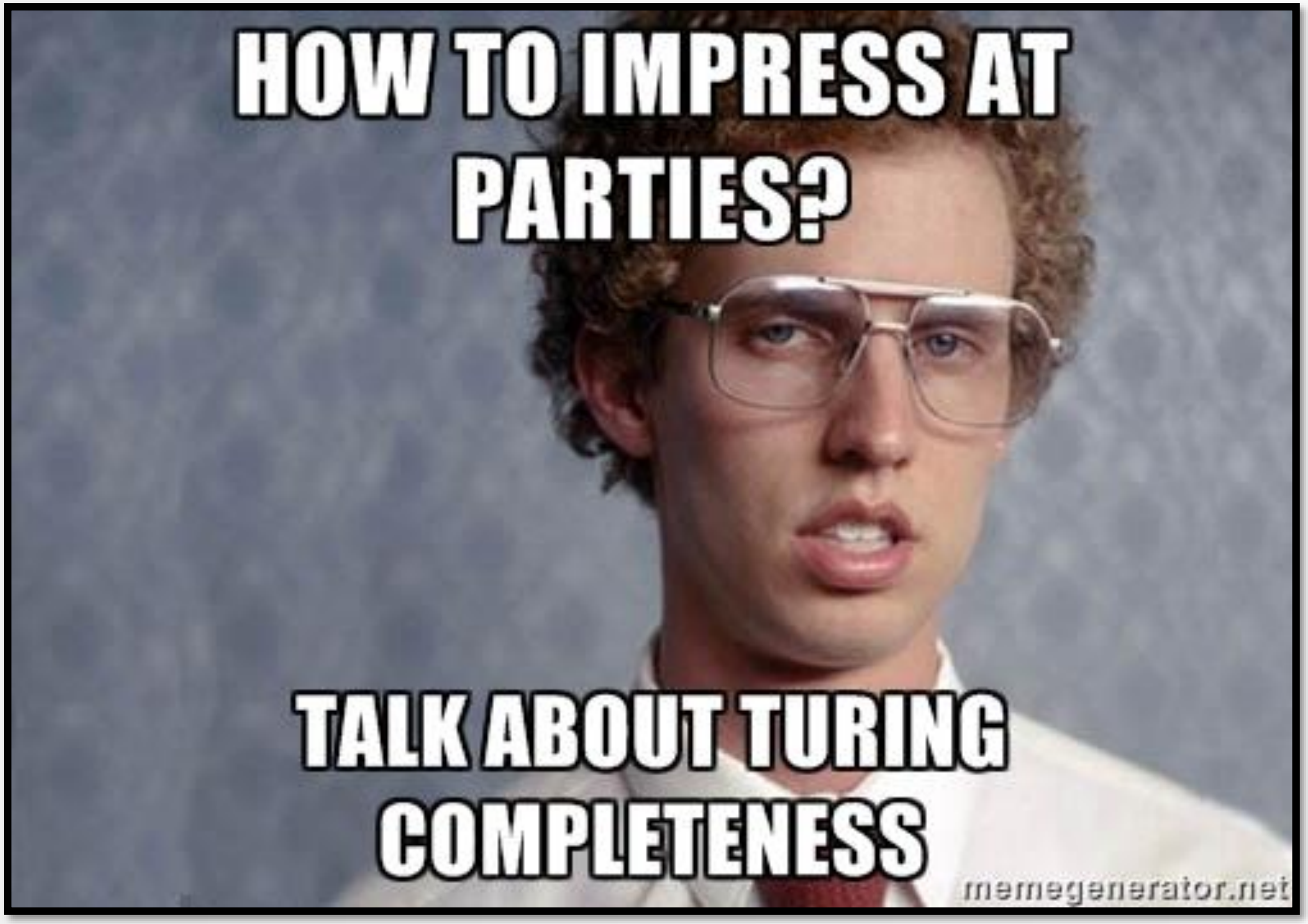
This is also SQL: Generating the Mandelbrot Set

```
-- Query from http://explainextended.com/2013/12/31/happy-new-year-5/
WITH RECURSIVE q(r, i, rx, ix, g) AS (
  SELECT r::DOUBLE PRECIS
        .0::DOUBLE PRECIS
  FROM generate_series(-6
  UNION ALL
  SELECT r, i, CASE WHEN
               CASE WHEN
  FROM q
  WHERE rx IS NOT NULL AN
)
SELECT array_to_string(ar
FROM (
  SELECT i, r, substring(
  FROM q
  GROUP BY i, r
) q
GROUP BY i
ORDER BY i
```



SQL:1999 is turing complete

SQL:1999 is turing complete



Seriously, what does that mean?

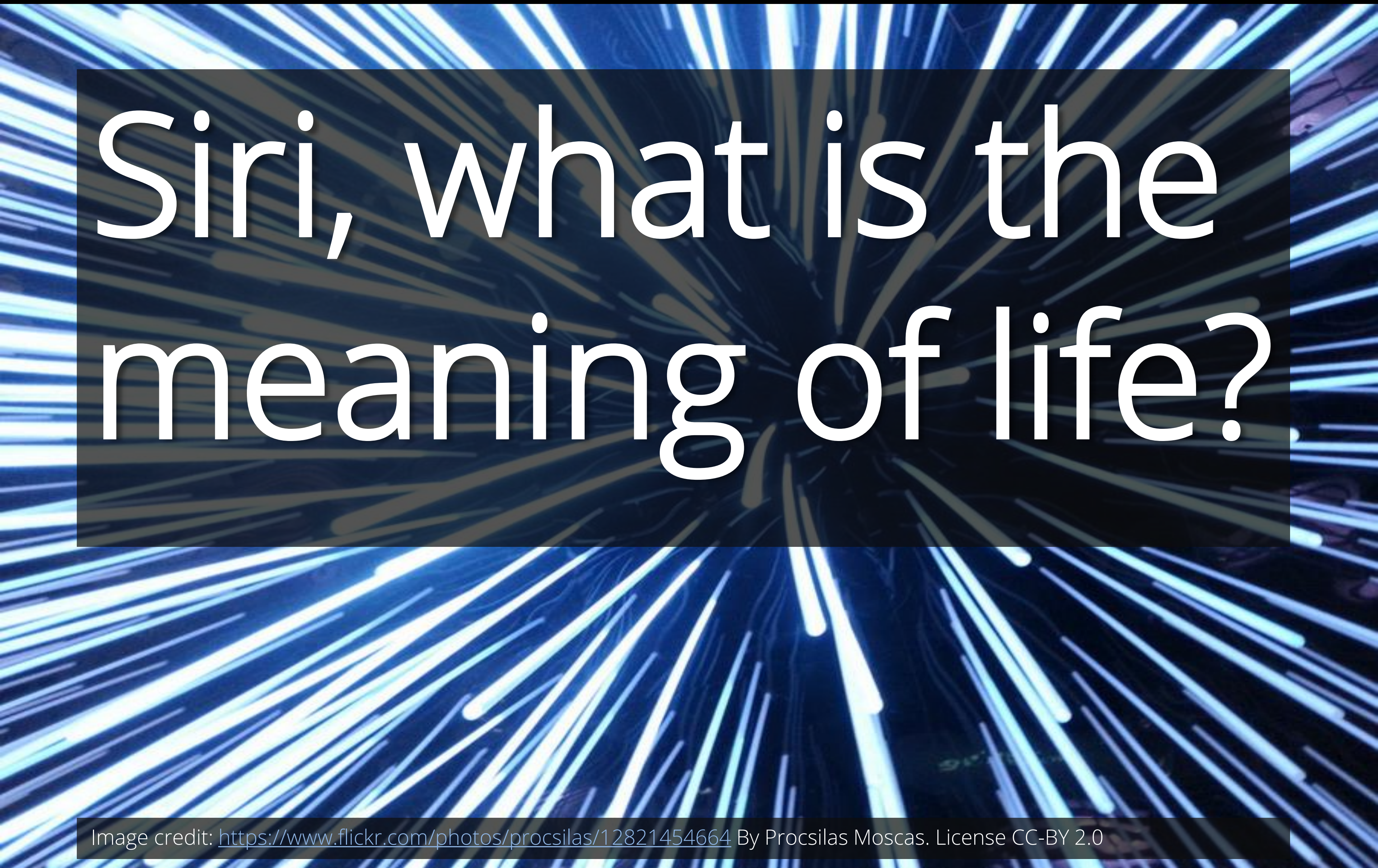
Any program can be written in SQL!

(although, no one’s that crazy)

The strength of a 4GL language

You tell the machine WHAT, not HOW

Which do you feel is more awesome? This?



Siri, what is the meaning of life?

Which do you feel is more awesome? Or this?

That’s why the company is called “Oracle”

What’s the problem with SQL?

What’s the problem with SQL? – SQL code

```
WITH RECURSIVE t(d) AS (
  SELECT DATE '2005-07-01'
  UNION ALL
  SELECT (d + INTERVAL '1 days')::DATE
  FROM t
  WHERE d < DATE '2005-07-31'
)
SELECT *
FROM t
```

What's the problem with SQL? – COBOL code

```
DATA DIVISION.
FILE SECTION.
FD   Sales-File.
01   Sales-Rec.
     88 End-Of-Sales-File          VALUE HIGH-VALUES.
     02  SF-Cust-Id                PIC X(5).
     02  SF-Cust-Name              PIC X(20).
     02  SF-Oil-Id.
         03  FILLER                PIC X.
             88 Essential-Oil      VALUE "1".
         03  SF-Oil-Name           PIC 99.
     02 SF-Unit-Size               PIC 99.
     02 SF-Units-Sold              PIC 999.
```

The syntax is awkward.

Why people don't like SQL

Declarative thinking is hard.

Reporting is «very easy» with SQL.

Bulk data processing is «very easy» with SQL.

Why people should like SQL

Ad-hoc analytics is «very easy» with SQL.

Why people should like SQL

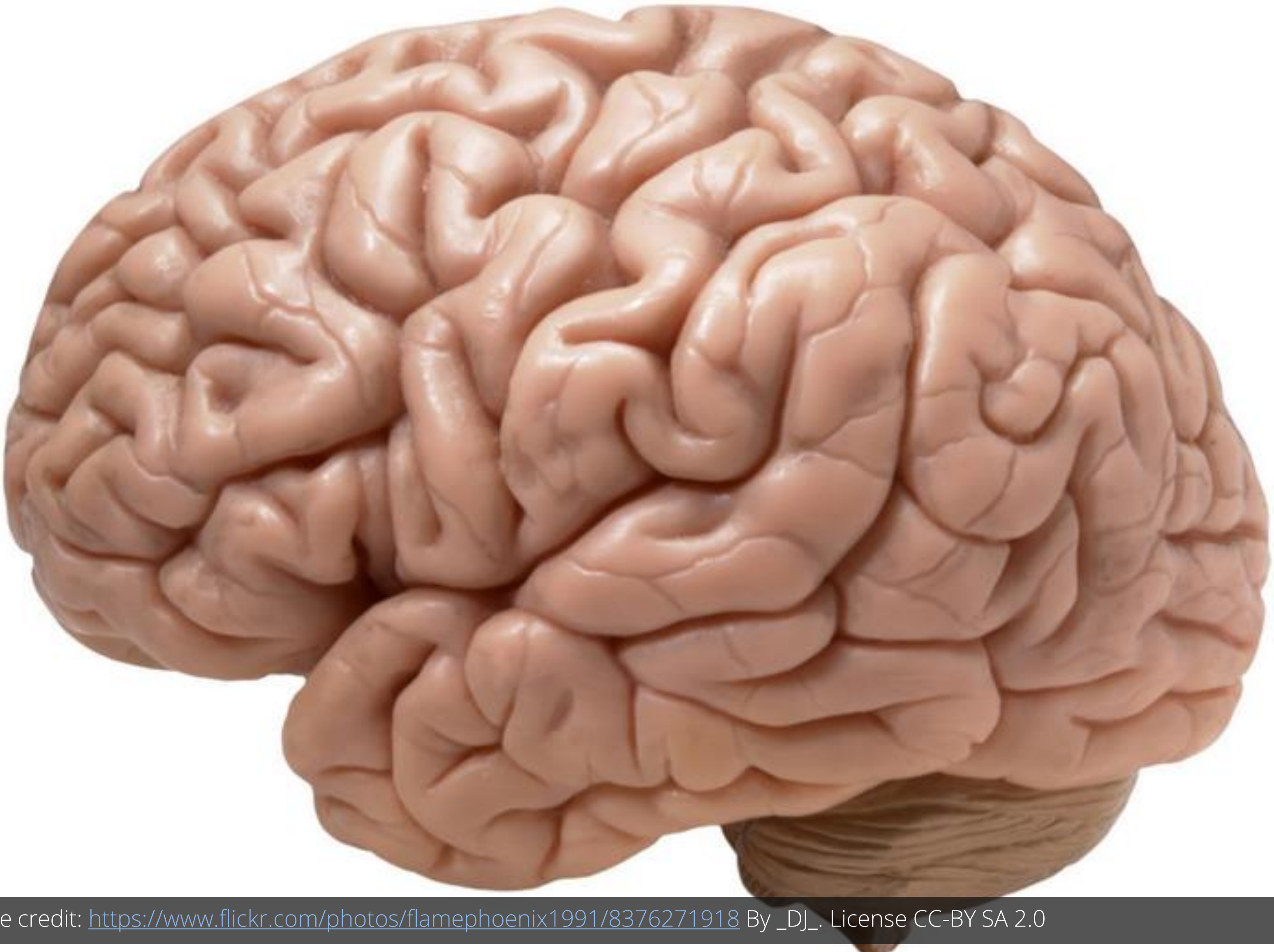
By «very easy» I mean hard.

But you don't have a choice.

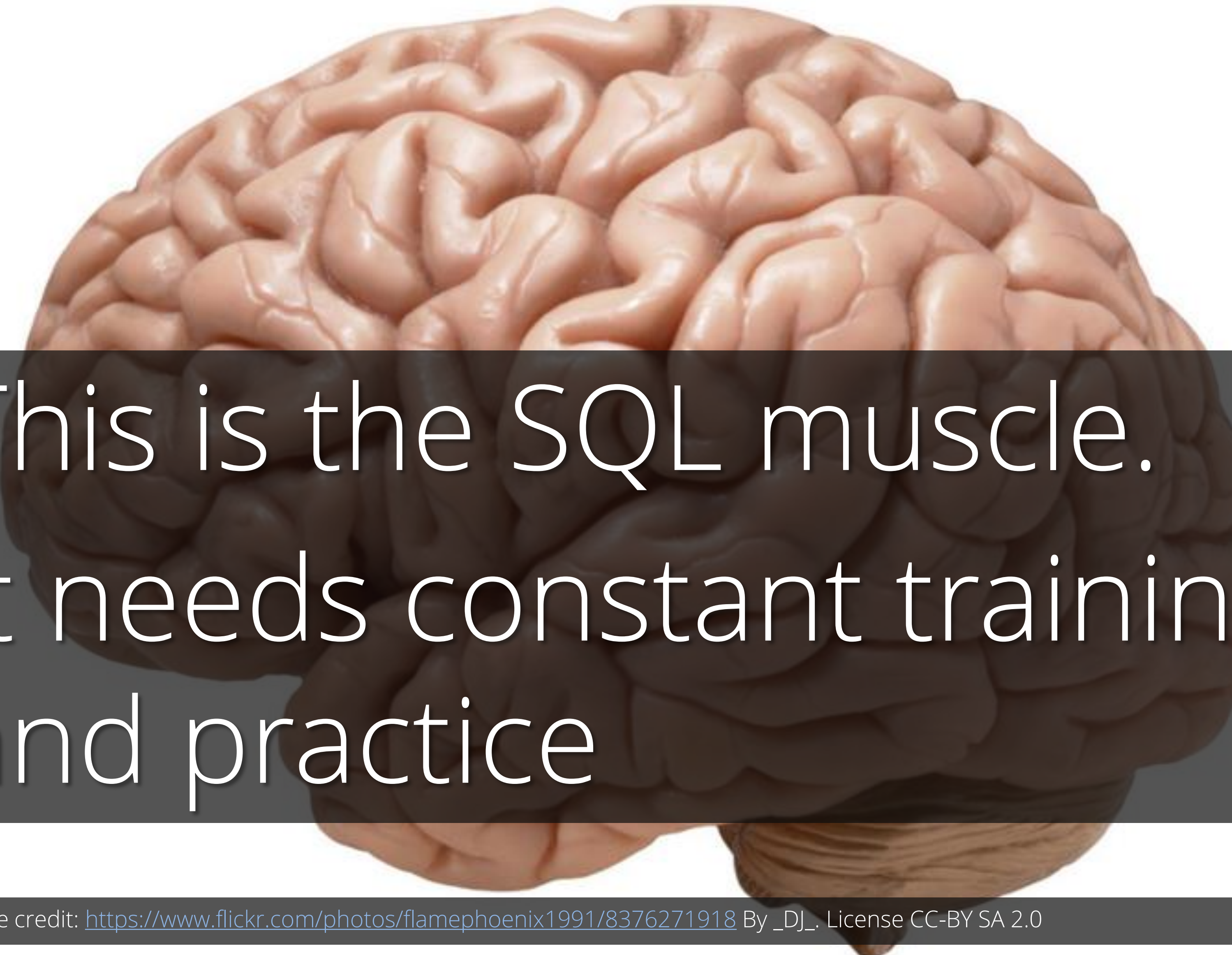
Winston Churchill on SQL

> "SQL is the worst form of database querying, except for all the other forms."

Remember this from this talk: The SQL muscle



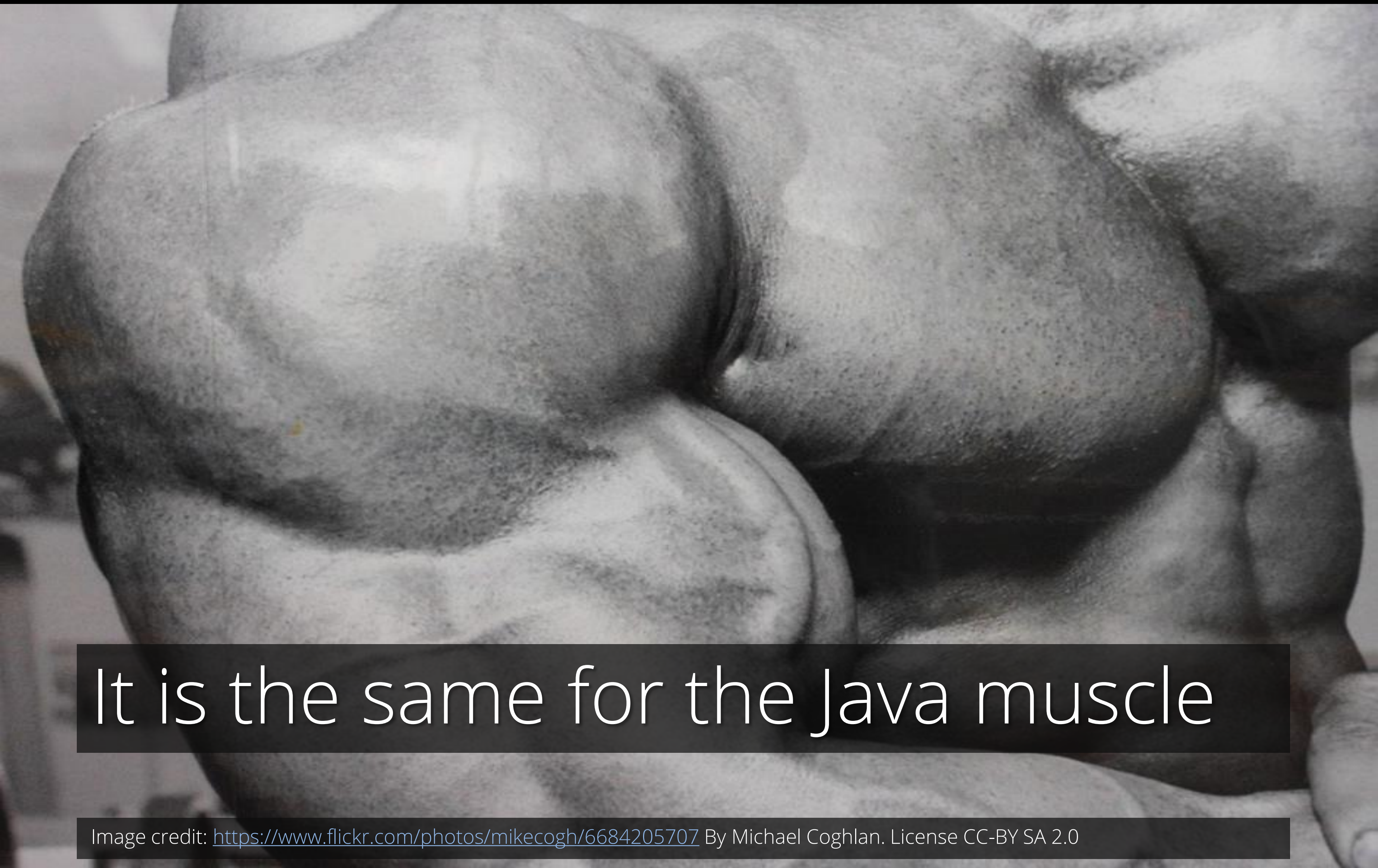
Remember this from this talk: The SQL muscle



This is the SQL muscle.

It needs constant training and practice

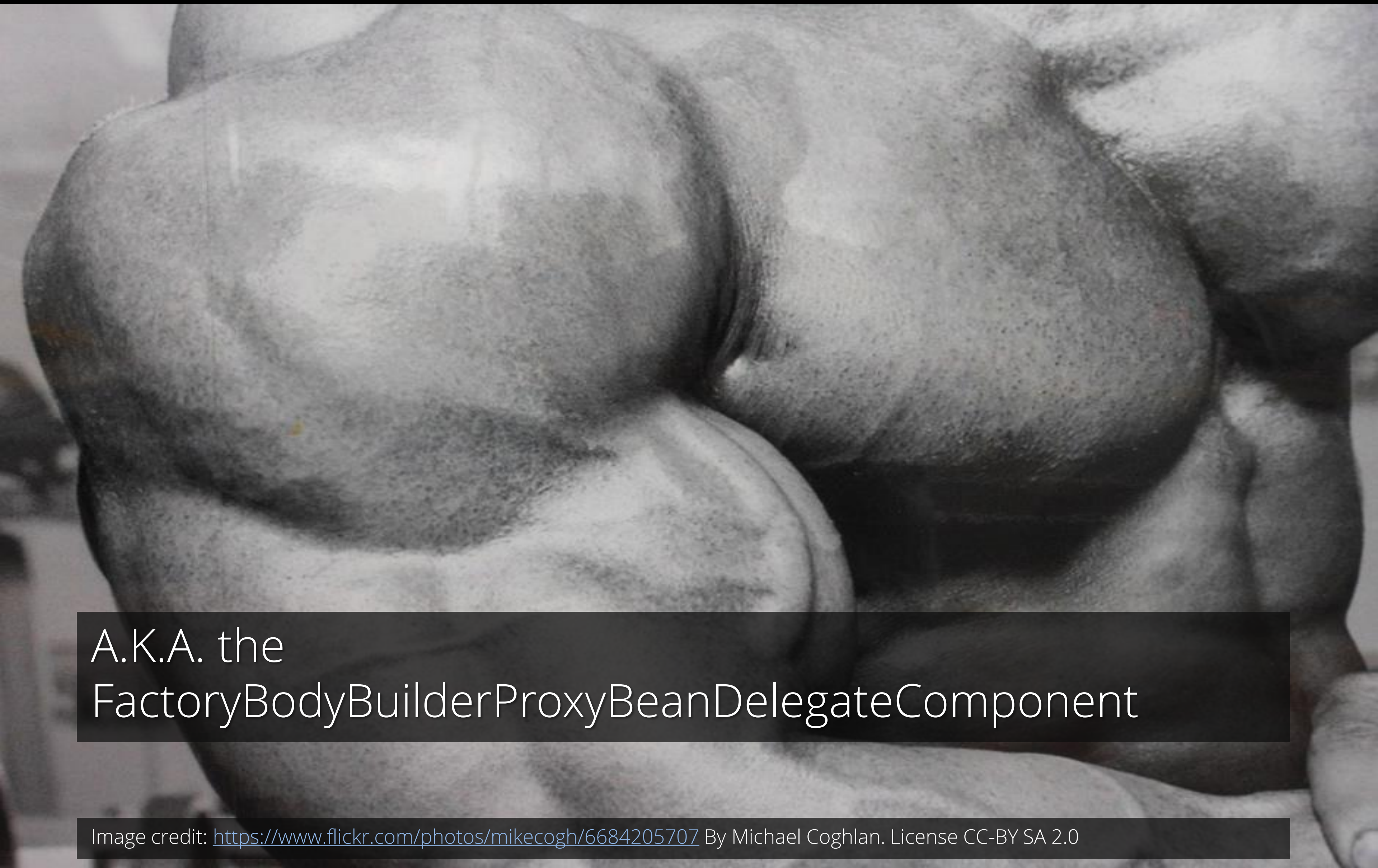
Remember this from this talk: The SQL muscle



It is the same for the Java muscle

Image credit: https://www.flickr.com/photos/mikecogh/6684205707 By Michael Coghlan. License CC-BY SA 2.0

Remember this from this talk: The SQL muscle



A.K.A. the FactoryBodyBuilderProxyBeanDelegateComponent

Image credit: https://www.flickr.com/photos/mikecogh/6684205707 By Michael Coghlan. License CC-BY SA 2.0

What you came here for

Enough bla bla

What you came here for...

10 SQL tricks to convince you SQL is awesome

1. Everything is a table
2. Data generation with recursive SQL
3. Running total calculations
4. Finding the length of a series
5. Finding the largest series with no gaps
6. The subset sum problem with SQL
7. Capping a running total
8. Time series pattern recognition
9. Pivoting and unpivoting
10. Abusing XML and JSON (don’t do this at home)

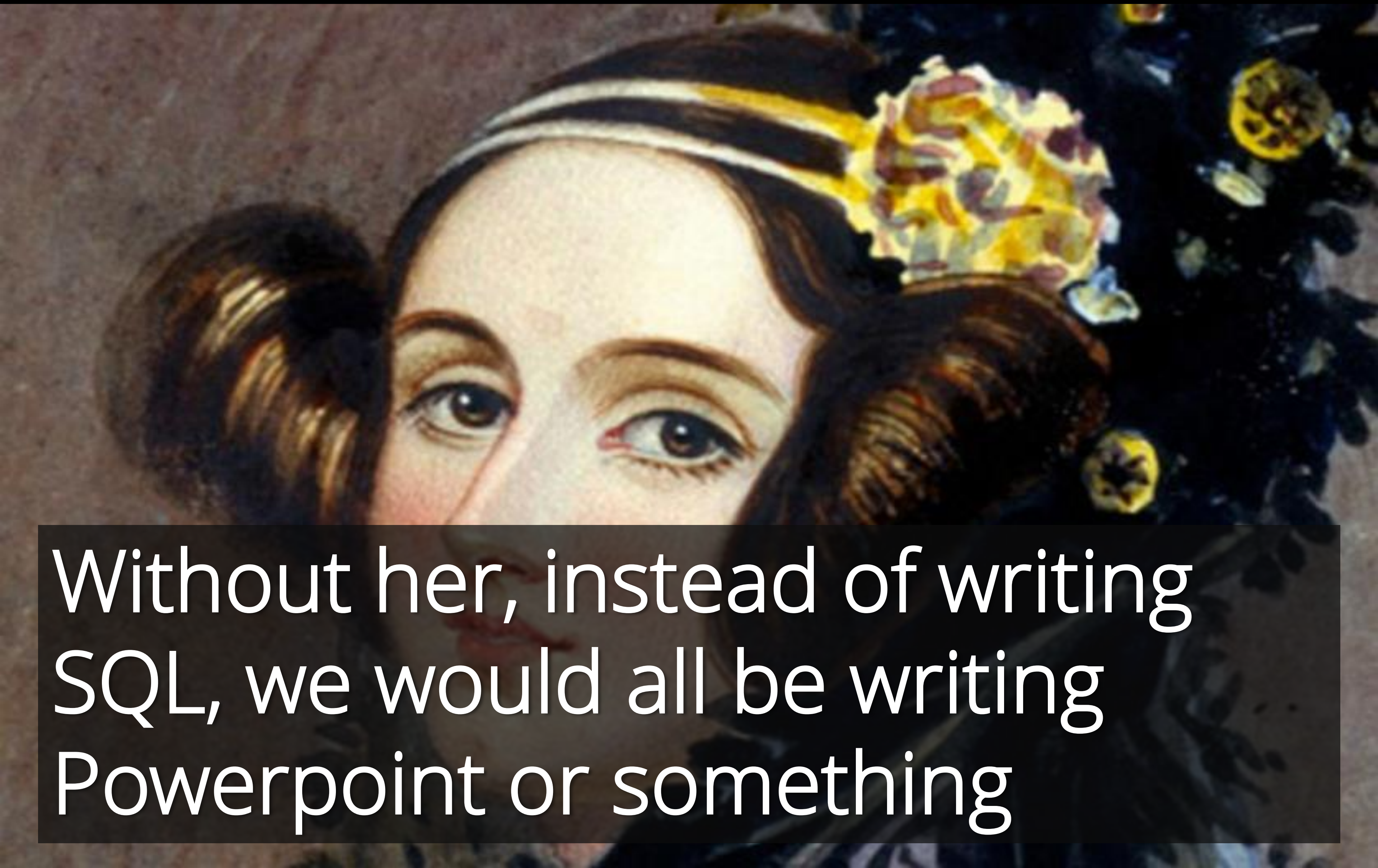
10 SQL tricks to convince you SQL is awesome

Are you really ready?

This presentation has roughly 5713 slides of SQL awesomeness!

Speaking of slides: Let’s thank our patron saint Ada Lovelace

Speaking of slides: Let’s thank our patron saint Ada Lovelace



Without her, instead of writing SQL, we would all be writing Powerpoint or something

1. Everything is a table

Most of you know this:

```
SELECT *
FROM person
```

Most of you know this:

```
SELECT *
FROM person
```

1. Everything is a table

Most of you know this:

```
SELECT *
FROM person
```

1. Everything is a table

Most of you also know this:

```
SELECT *
FROM (
  SELECT *
  FROM person
) AS t -- "derived table"
```

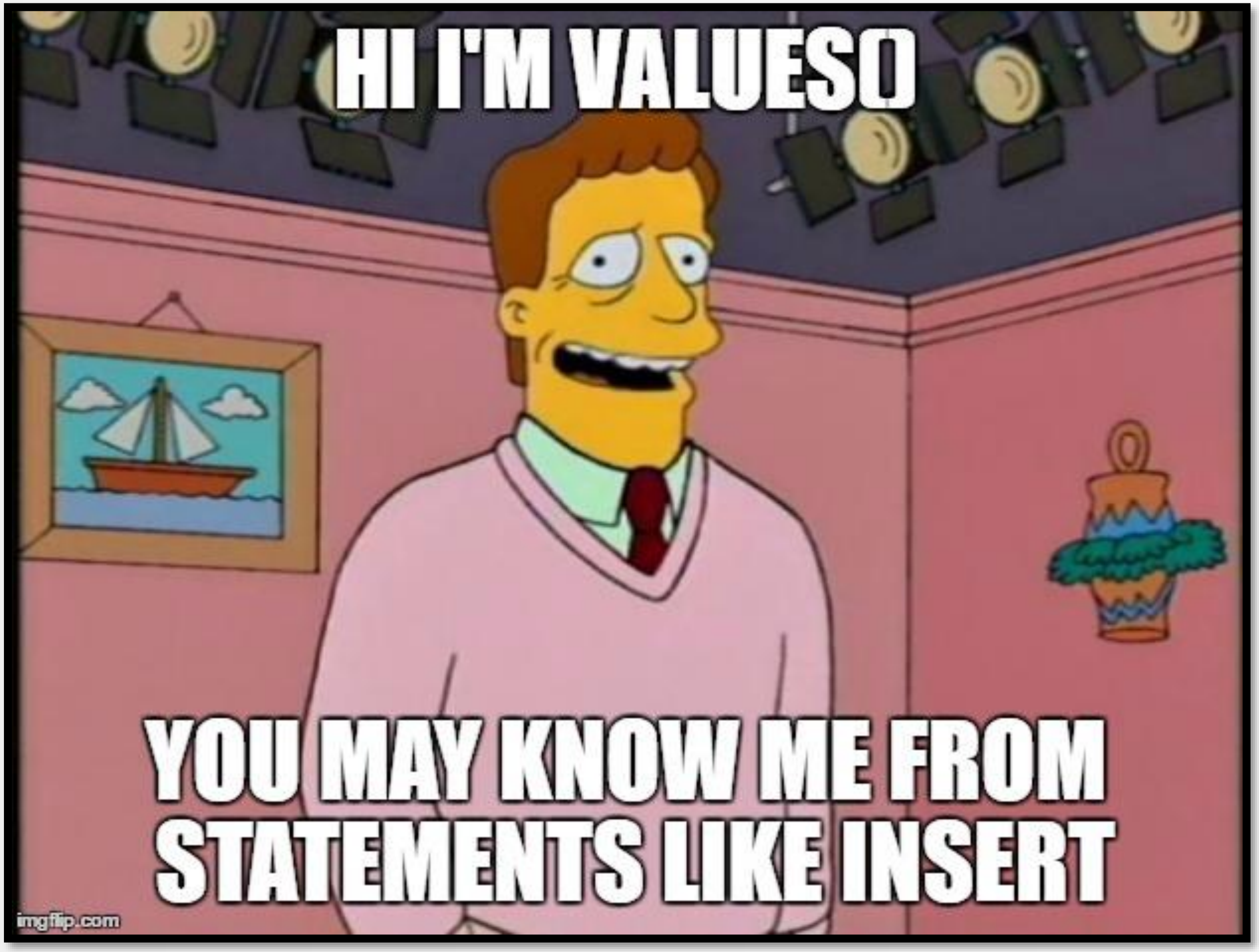
1. Everything is a table

But did you know this?

```
SELECT *
FROM (
  -- "values constructor"
  VALUES (1, 'a'), (2, 'b')
) t(a, b) -- "derived column list"
```

	a integer	b text
1	1	a
2	2	b

1. Everything is a table



1. Everything is a table

But did you know this?

```
SELECT *
FROM (
 SELECT 1 AS a, 'a' AS b FROM dual
 UNION ALL
 SELECT 2,      'b'      FROM dual
) t
```

	a integer	b text
1	1	a
2	2	b

1. Everything is a table

Or this?

	substring text
1	bcd

```
SELECT *
FROM substring('abcde', 2, 3)
```

1. Everything is a table – Compare it to Java 8

```
TABLE          : Stream<Tuple<..>>
SELECT         : map()
DISTINCT       : distinct()
JOIN           : flatMap()
WHERE / HAVING : filter()
GROUP BY       : collect()
ORDER BY       : sorted()
UNION ALL      : concat()
```

See:
http://blog.jooq.org/2015/08/13/common-sql-clauses-and-their-equivalents-in-java-8-streams/

1. Everything is a table – Compare it to Java 8

Better Streams:

https://github.com/jOOQ/jOOL

1. Everything is a table – Compare it to Java 8

```
Seq.seq(persons)
    .collect(
        count(),
        max(Person::getAge),
        min(Person::getHeight),
        avg(Person::getWeight)
    );
// (3, Optional[35],
//   Optional[1.69], Optional[70.0])
```

2. Data Generation with Recursive SQL

Common Table Expressions

The only way to declare variables in SQL

2. Data Generation with Recursive SQL

```sql
-- Table variables
WITH
  t1(v1, v2) AS (SELECT 1, 2),
  t2(w1, w2) AS (
    SELECT v1 * 2, v2 * 2
    FROM t1
  )
SELECT *
FROM t1, t2
```

	v1 integer	v2 integer	w1 integer	w2 integer
1	1	2	2	4

2. Data Generation with Recursive SQL

```
WITH RECURSIVE t(v) AS (
  SELECT 1      -- Seed Row
  UNION ALL
  SELECT v + 1 -- Recursion
  FROM t
)
SELECT v
FROM t
LIMIT 5
```

	v integer
1	1
2	2
3	3
4	4
5	5

2. Data Generation with Recursive SQL

2. D

```sql
WITH RECURSIVE t(v) AS (
  SELECT 1       -- Seed Row
  UNION ALL
  SELECT v + 1 -- Recursion
  FROM t
)
SELECT v
FROM t
LIMIT 5
```

	v intege
1	
2	
3	
4	
5	

Copyright (c) 2009-2016 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0

2. D

```sql
WITH RECURSIVE t(v) AS (
  SELECT 1       -- Seed Row
  UNION ALL
  SELECT v + 1 -- Recursion
  FROM t
)
SELECT v
FROM t
LIMIT 5
```

	v in
1	
2	
3	
4	
5	

Copyright (c) 2009-2016 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0
Copyright (c) 2009-2016 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0

2. D

```sql
WITH RECURSIVE t(v) AS (
  SELECT 1       -- Seed Row
  UNION ALL
  SELECT v + 1 -- Recursion
  FROM t
)
SELECT v
FROM t
LIMIT 5
```

1
2
3
4
5

Copyright (c) 2009-2016 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0
Copyright (c) 2009-2016 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0
Copyright (c) 2009-2016 by Data Geekery GmbH. Slides licensed under CC BY SA 3.0

Credits for this lame Powerpoint joke:

Hadi Hariri from JetBrains

2. Data Generation with Recursive SQL

```sql
WITH RECURSIVE t(v) AS (
  SELECT 1       -- Seed Row
  UNION ALL
  SELECT v + 1   -- Recursion
  FROM t
)
SELECT v
FROM t
LIMIT 5
```

	v integer
1	1
2	2
3	3
4	4
5	5

2. Data Generation with Recursive SQL

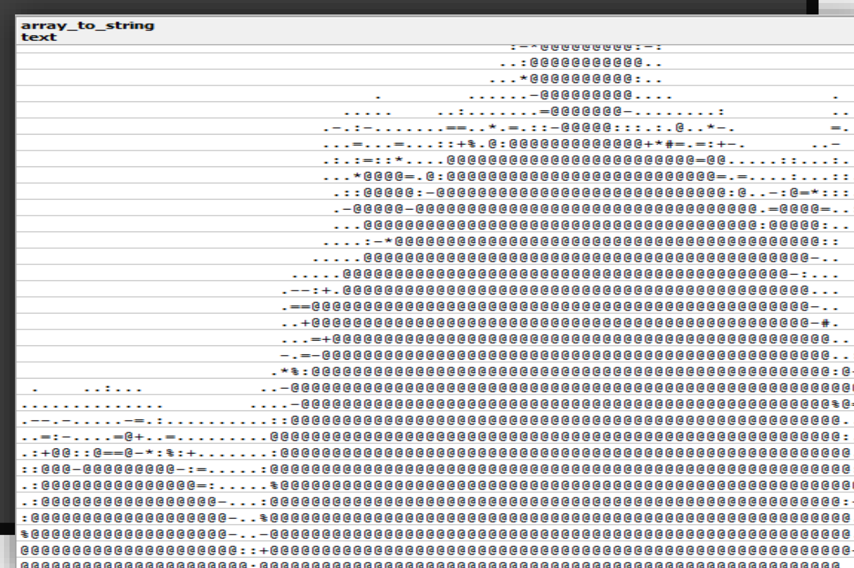
```
SELECT level AS v
FROM dual
CONNECT BY level <= 5
```

	v integer
1	1
2	2
3	3
4	4
5	5

ORACLE

Remember?

```
-- Query from http://explainextended.com/2013/12/31/happy-new-year-5/
WITH RECURSIVE q(r, i, rx, ix, g) AS (
  SELECT r::DOUBLE PRECISION * 0.02, i::DOUBLE PRECISION * 0.02,
        .0::DOUBLE PRECISION      , .0::DOUBLE PRECISION, 0
  FROM generate_series(-60, 20) r, generate_series(-50, 50) i
  UNION ALL
  SELECT r, i, CASE WHEN abs(rx * rx + ix * ix) <= 2 THEN rx * rx - ix * ix END + r,
               CASE WHEN abs(rx * rx + ix * ix) <= 2 THEN 2 * rx * ix END + i, g + 1
  FROM q
  WHERE rx IS NOT NULL AND g < 99
)
SELECT array_to_string(array_agg(s ORDER BY r),
FROM (
  SELECT i, r, substring(' .:-=+*#%@', max(g) /
  FROM q
  GROUP BY i, r
) q
GROUP BY i
ORDER BY i
```



2. Data Generation with Recursive SQL

Applications:

1. Iterate from 1 to 10
2. Generate all dates in July 2016
3. Generating graphs (stay tuned!)

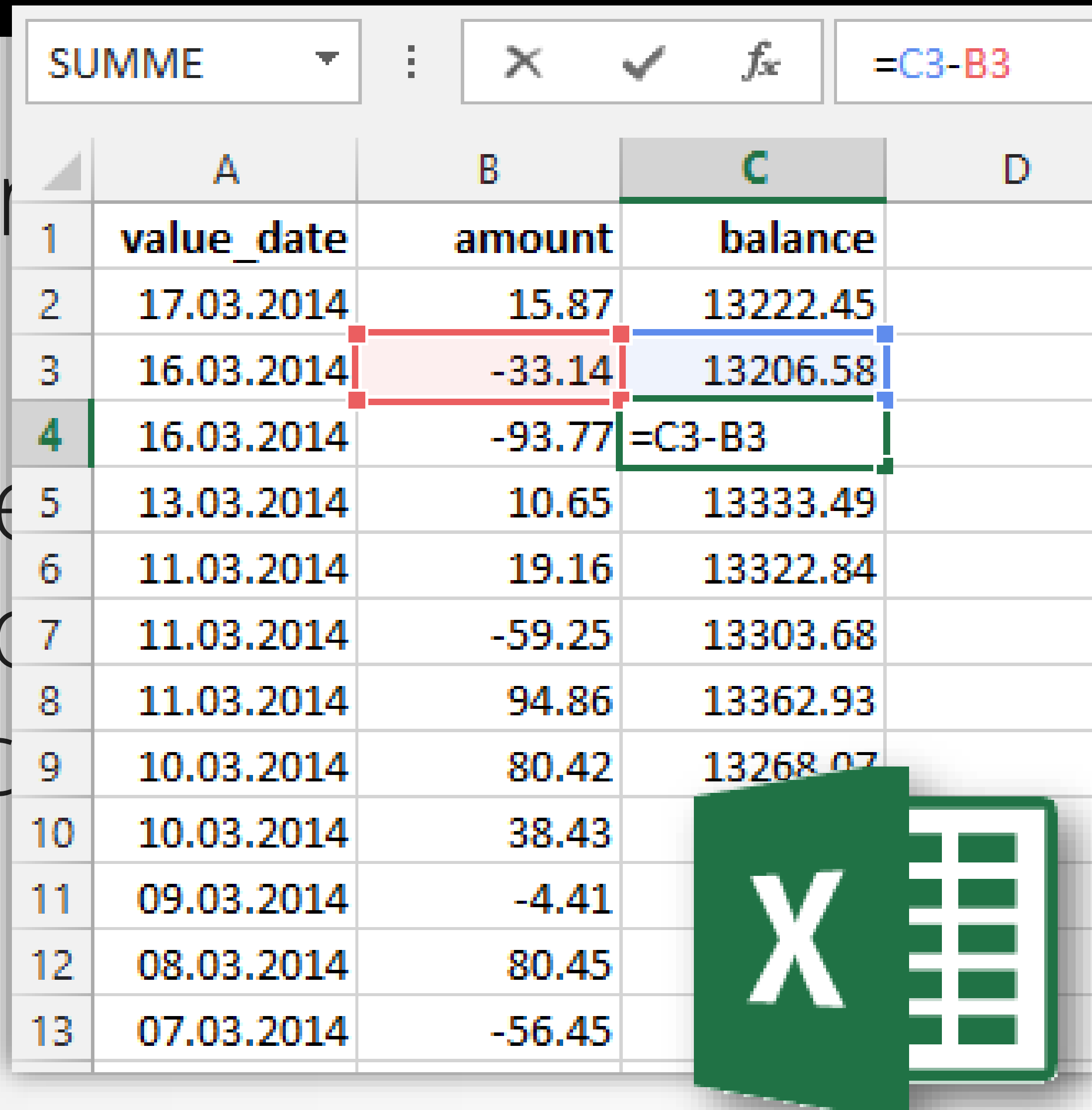
What is a running total?

Ask your project manager to give you a crash course about the awesome Microsoft Excel!

3. Running total calculations

What is a runn

Ask your proje
you a crash co
awesome Mic


SUMMME =C3-B3

	A	B	C	D
1	value_date	amount	balance	
2	17.03.2014	15.87	13222.45	
3	16.03.2014	-33.14	13206.58	
4	16.03.2014	-93.77	=C3-B3	
5	13.03.2014	10.65	13333.49	
6	11.03.2014	19.16	13322.84	
7	11.03.2014	-59.25	13303.68	
8	11.03.2014	94.86	13362.93	
9	10.03.2014	80.42	13268.07	
10	10.03.2014	38.43		
11	09.03.2014	-4.41		
12	08.03.2014	80.45		
13	07.03.2014	-56.45		

3. Running total calculations

But first, a little theory about window functions

There was SQL before window functions and there was SQL after window functions.

What are window functions?

```
-- Aggregations / rankings on a subset of
-- rows relative to the current row being
-- transformed by SELECT
function(...) OVER (
  PARTITION BY ...
  ORDER BY ...
  ROWS BETWEEN ... AND ...
)
```

3. Running total calculations

```
-- Aggregations / rankings on a subset of
-- rows relative to the current row being
-- transformed by SELECT
function(...) OVER (
  PARTITION BY length
  ORDER BY ...
  ROWS BETWEEN ... AND ...
)
```

row_number bigint	title character varying(255)	length smallint
1	ALIEN CENTER	46
2	IRON MOON	46
3	KWAI HOMEWARD	46
4	LABYRINTH LEAGUE	46
5	RIDGEMONT SUBMARINE	46
1	DIVORCE SHINING	47
2	DOWNHILL ENOUGH	47
3	HALLOWEEN NUTS	47
4	HANOVER GALAXY	47
5	HAWK CHILL	47
6	SHANGHAI TYCOON	47
7	SUSPECTS QUILLS	47
1	ACE GOLDFINGER	48

3. Running total calculations

```
-- Aggregations / rankings on a subset of
-- rows relative to the current row being
-- transformed by SELECT
function(...) OVER (
  PARTITION BY length
  ORDER BY ...
  ROWS BETWEEN ... AND ...
)
```

row_number bigint	title character varying(255)	length smallint
1	ALIEN CENTER	46
2	IRON MOON	46
3	KWAI HOMEWARD	46
4	LABYRINTH LEAGUE	46
5	RIDGEMONT SUBMARINE	46
1	DIVORCE SHINING	47
2	DOWNHILL ENOUGH	47
3	HALLOWEEN NUTS	47
4	HANOVER GALAXY	47
5	HAWK CHILL	47
6	SHANGHAI TYCOON	47
7	SUSPECTS QUILLS	47
1	ACE GOLDFINGER	48

3. Running total calculations

```sql
-- Aggregations / rankings on a subset of
-- rows relative to the current row being
-- transformed by SELECT
function(...) OVER (
  PARTITION BY ...
  ORDER BY title
  ROWS BETWEEN ... AND ...
)
```

row_number bigint	title character varying(255)	length smallint
1	ALIEN CENTER	46
2	IRON MOON	46
3	KWAI HOMEWARD	46
4	LABYRINTH LEAGUE	46
5	RIDGEMONT SUBMARINE	46
1	DIVORCE SHINING	47
2	DOWNHILL ENOUGH	47
3	HALLOWEEN NUTS	47
4	HANOVER GALAXY	47
5	HAWK CHILL	47
6	SHANGHAI TYCOON	47
7	SUSPECTS QUILLS	47
1	ACE GOLDFINGER	48

3. Running total calculations

```
-- Aggregations / rankings on a subset of
-- rows relative to the current row being
-- transformed by SELECT
function(...) OVER (
  PARTITION BY ...
  ORDER BY title
  ROWS BETWEEN ... AND ...
)
```

row_number bigint	title character varying(255)	length smallint
1	ALIEN CENTER	46
2	IRON MOON	46
3	KWAI HOMEWARD	46
4	LABYRINTH LEAGUE	46
5	RIDGEMONT SUBMARINE	46
1	DIVORCE SHINING	47
2	DOWNHILL ENOUGH	47
3	HALLOWEEN NUTS	47
4	HANOVER GALAXY	47
5	HAWK CHILL	47
6	SHANGHAI TYCOON	47
7	SUSPECTS QUILLS	47
1	ACE GOLDFINGER	48

3. Running total calculations

```
-- Aggregations / rankings on a subset of
-- rows relative to the current row being
-- transformed by SELECT
row_number( ) OVER (
  PARTITION BY ...
  ORDER BY title
  ROWS BETWEEN ... AND ...
)
```

row_number bigint	title character varying(255)	length smallint
1	ALIEN CENTER	46
2	IRON MOON	46
3	KWAI HOMEWARD	46
4	LABYRINTH LEAGUE	46
5	RIDGEMONT SUBMARINE	46
1	DIVORCE SHINING	47
2	DOWNHILL ENOUGH	47
3	HALLOWEEN NUTS	47
4	HANOVER GALAXY	47
5	HAWK CHILL	47
6	SHANGHAI TYCOON	47
7	SUSPECTS QUILLS	47
1	ACE GOLDFINGER	48

Let this settle a bit

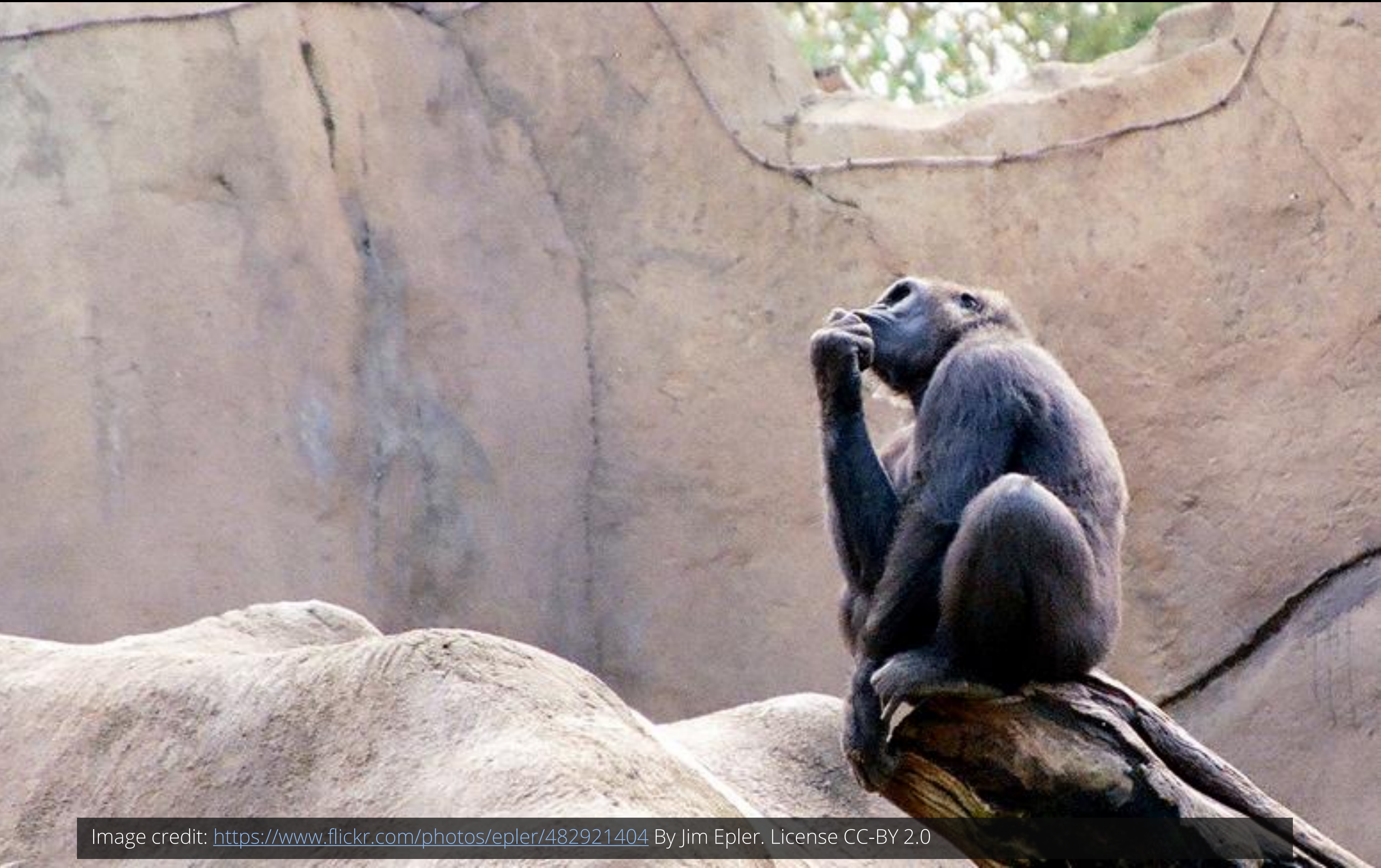


Image credit: https://www.flickr.com/photos/epler/482921404 By Jim Epler. License CC-BY 2.0

Let this settle a bit

Window functions are aggregations / rankings on a subset of rows relative to the current row being transformed by SELECT

3. Running total calculations

This is the data in the database table

```
| ID   | VALUE_DATE | AMOUNT |
|------|------------|--------|
| 9997 | 2014-03-18 |  99.17 |
| 9981 | 2014-03-16 |  71.44 |
| 9979 | 2014-03-16 | -94.60 |
| 9977 | 2014-03-16 |  -6.96 |
| 9971 | 2014-03-15 | -65.95 |
```

3. Running total calculations

This is what we want to calculate

ID	VALUE_DATE	AMOUNT	BALANCE
9997	2014-03-18	99.17	19985.81
9981	2014-03-16	71.44	19886.64
9979	2014-03-16	-94.60	19815.20
9977	2014-03-16	-6.96	19909.80
9971	2014-03-15	-65.95	19916.76

3. Running total calculations

This is how we calculate it

```
| ID   | VALUE_DATE | AMOUNT   |     BALANCE |
|------|------------|----------|-------------|
| 9997 | 2014-03-18  -(99.17)|   +19985.81 |
| 9981 | 2014-03-16  -(71.44)|    19886.64 |
| 9979 | 2014-03-16 -(-94.60)|    19815.20 |
| 9977 | 2014-03-16 |   -6.96 |   =19909.80 |
| 9971 | 2014-03-15 |  -65.95 |    19916.76 |
```

```sql
SUM(t.amount) OVER (
  PARTITION BY t.account_id
  ORDER BY     t.value_date DESC,
               t.id         DESC
  ROWS BETWEEN UNBOUNDED PRECEDING
       AND     1         PRECEDING
)
```

```sql
SUM(t.amount) OVER (
  PARTITION BY t.account_id
  ORDER BY     t.value_date DESC,
               t.id         DESC
  ROWS BETWEEN UNBOUNDED PRECEDING
       AND     1          PRECEDING
)
```

```sql
SUM(t.amount) OVER (
  PARTITION BY t.account_id
  ORDER BY     t.value_date DESC,
               t.id         DESC
  ROWS BETWEEN UNBOUNDED PRECEDING
       AND     1          PRECEDING
)
```

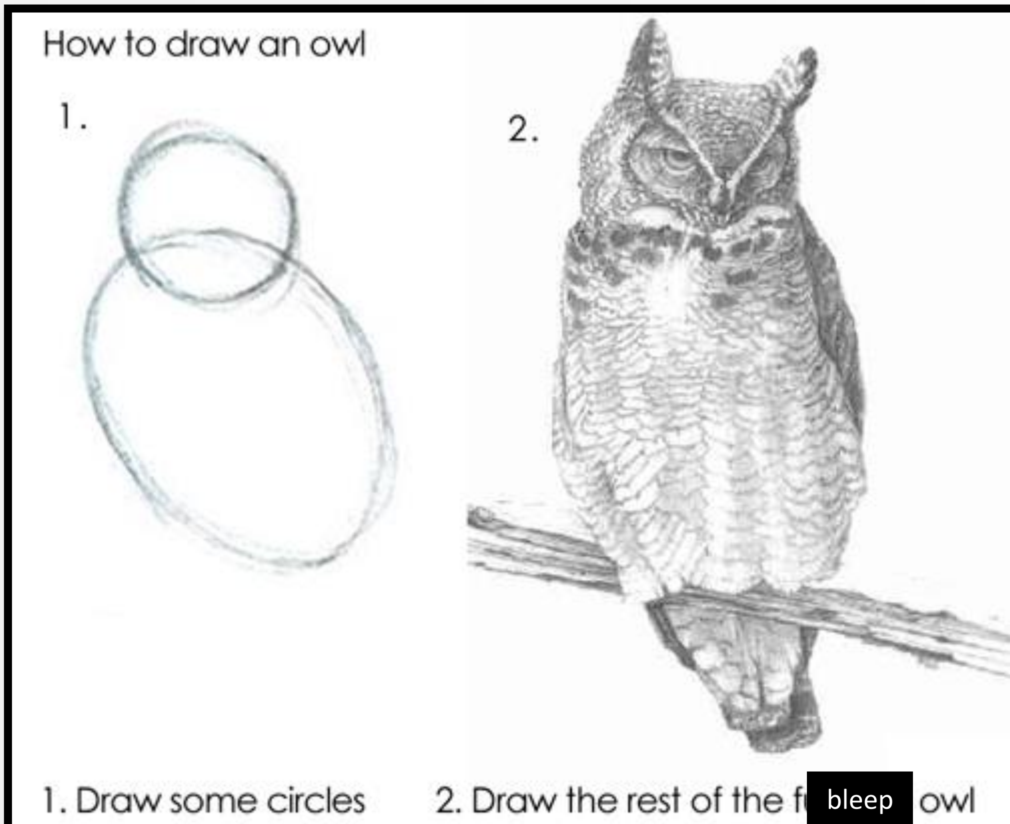
```sql
SUM(t.amount) OVER (
  PARTITION BY t.account_id
  ORDER BY     t.value_date DESC,
               t.id         DESC
  ROWS BETWEEN UNBOUNDED PRECEDING
       AND     1         PRECEDING
)
```

Now we have the tool set.

Remember these two advanced SQL features:

1. (Recursive) common table expressions
2. Window functions

Now we have the tool set. Are you ready?



Don’t worry.

Don’t worry if this is how you feel:

Experience comes with practice!

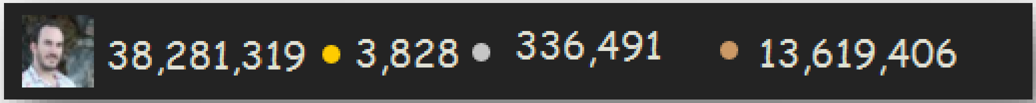
4. Finding the largest series with no gaps

stackoverflow

Enthusiast	Visit the site each day for 30 consecutive days.
Fanatic	Visit the site each day for 100 consecutive days.

4. Finding the largest series with no gaps

Enthusiast	Visit the site each day for 30 consecutive days.
Fanatic	Visit the site each day for 100 consecutive days.



4. Finding the largest series with no gaps

```
| LOGIN_TIME          |
|---------------------|
| 2014-03-18 05:37:13 |
| 2014-03-16 08:31:47 |
| 2014-03-16 06:11:17 |
| 2014-03-16 05:59:33 |
| 2014-03-15 11:17:28 |
| 2014-03-15 10:00:11 |
| 2014-03-15 07:45:27 |
| 2014-03-15 07:42:19 |
| 2014-03-14 09:38:12 |
```

4. Finding the largest series with no gaps

```
| LOGIN_DATE |
|------------|
| 2014-03-18 |
| 2014-03-16 |
| 2014-03-15 |
| 2014-03-14 |
```

4. Finding the largest series with no gaps

Easy…

```
SELECT DISTINCT
  cast(login_time AS DATE) AS login_date
FROM logins
WHERE user_id = :user_id
```

4. Finding the largest series with no gaps

```
| LOGIN_DATE | RN |
|------------|----|
| 2014-03-18 |  4 |
| 2014-03-16 |  3 |
| 2014-03-15 |  2 |
| 2014-03-14 |  1 |
```

4. Finding the largest series with no gaps

Still easy…

```
SELECT
  login_date,
  row_number() OVER (ORDER BY login_date)
FROM login_dates
```

4. Finding the largest series with no gaps

Now, what happens if we subtract...?

```
SELECT
  login_date -
  row_number() OVER (ORDER BY login_date)
FROM login_dates
```

4. Finding the largest series with no gaps

```
| LOGIN_DATE | RN | GRP        |
|------------|----|------------|
| 2014-03-18 |  4 | 2014-03-14 |
| 2014-03-16 |  3 | 2014-03-13 |
| 2014-03-15 |  2 | 2014-03-13 |
| 2014-03-14 |  1 | 2014-03-13 |
```

4. Finding the largest series with no gaps

Gap here

	RN	GRP
2014-03-18	4	2014-03-14
2014-03-16	3	2014-03-13
2014-03-15	2	2014-03-13
2014-03-14	1	2014

Gap here

4. Finding the largest series with no gaps

4. Finding the largest series with no gaps

Easy explanation:

1. ROW_NUMBER() never has gaps
2. Our data, however, does

4. Finding the largest series with no gaps

So, just group by this difference!

```
SELECT
  min(login_date), max(login_date),
  max(login_date) -
  min(login_date) + 1 AS length
FROM login_date_groups
GROUP BY grp
ORDER BY length DESC
```

4. Finding the largest series with no gaps

```
| MIN          | MAX          | LENGTH |
|--------------|--------------|--------|
| 2014-03-14   | 2014-03-16   |      3 |
| 2014-03-18   | 2014-03-18   |      1 |
```

4. Finding the largest series with no gaps

```
WITH
  login_dates AS (
    SELECT DISTINCT cast(login_time AS DATE) login_date
    FROM logins WHERE user_id = :user_id
  ),
  login_date_groups AS (
    SELECT
      login_date,
      login_date - row_number() OVER (ORDER BY login_date) AS grp
    FROM login_dates
  )
SELECT
  min(login_date), max(login_date),
  max(login_date) - min(login_date) + 1 AS length
FROM login_date_groups
GROUP BY grp
ORDER BY length DESC
```

OK?

5. Finding the length of a series

```
| ID   | VALUE_DATE | AMOUNT |
|------|------------|--------|
| 9997 | 2014-03-18 |  99.17 |
| 9981 | 2014-03-16 |  71.44 |
| 9979 | 2014-03-16 | -94.60 |
| 9977 | 2014-03-16 |  -6.96 |
| 9971 | 2014-03-15 | -65.95 |
| 9964 | 2014-03-15 |  15.13 |
| 9962 | 2014-03-15 |  17.47 |
| 9960 | 2014-03-15 |  -3.55 |
| 9959 | 2014-03-14 |  32.00 |
```

5. Finding the length of a series

```
| ID   | VALUE_DATE | AMOUNT |       LENGTH |
|------|------------|--------|------------|
| 9997 | 2014-03-18 |  99.17 |            2 |
| 9981 | 2014-03-16 |  71.44 |            2 |
| 9979 | 2014-03-16 | -94.60 |            3 |
| 9977 | 2014-03-16 |  -6.96 |            3 |
| 9971 | 2014-03-15 | -65.95 |            3 |
| 9964 | 2014-03-15 |  15.13 |            2 |
| 9962 | 2014-03-15 |  17.47 |            2 |
| 9960 | 2014-03-15 |  -3.55 |            1 |
| 9959 | 2014-03-14 |  32.00 |            1 |
```

5. Finding the length of a series

ID	VALUE_DATE	AMOUNT	LENGTH
9997	2014-03-18	+99.17	2
9981	2014-03-16	+71.44	2
9979	2014-03-16	-94.60	3
9977	2014-03-16	-6.96	3
9971	2014-03-15	-65.95	3
9964	2014-03-15	15.13	2
9962	2014-03-15	17.47	2
9960	2014-03-15	-3.55	1
9959	2014-03-14	32.00	1

5. Finding the length of a series

ID	VALUE_DATE	AMOUNT	LENGTH
9997	2014-03-18	99.17	2
9981	2014-03-16	71.44	2
9979	2014-03-16	-94.60	3
9977	2014-03-16	-6.96	3
9971	2014-03-15	-65.95	3
9964	2014-03-15	15.13	2
9962	2014-03-15	17.47	2
9960	2014-03-15	-3.55	1
9959	2014-03-14	32.00	1

5. Finding the length of a series

ID	VALUE_DATE	AMOUNT	LENGTH
9997	2014-03-18	99.17	2
9981	2014-03-16	71.44	2
9979	2014-03-16	-94.60	3
9977	2014-03-16	-6.96	3
9971	2014-03-15	-65.95	3
9964	2014-03-15	+15.13	2
9962	2014-03-15	+17.47	2
9960	2014-03-15	-3.55	1
9959	2014-03-14	32.00	1

5. Finding the length of a series

ID	VALUE_DATE	AMOUNT	LENGTH
9997	2014-03-18	99.17	2
9981	2014-03-16	71.44	2
9979	2014-03-16	-94.60	3
9977	2014-03-16	-6.96	3
9971	2014-03-15	-65.95	3
9964	2014-03-15	15.13	2
9962	2014-03-15	17.47	2
9960	2014-03-15	-3.55	1
9959	2014-03-14	32.00	1

5. Finding the length of a series

ID	VALUE_DATE	AMOUNT	LENGTH
9997	2014-03-18	99.17	2
9981	2014-03-16	71.44	2
9979	2014-03-16	-94.60	3
9977	2014-03-16	-6.96	3
9971	2014-03-15	-65.95	3
9964	2014-03-15	15.13	2
9962	2014-03-15	17.47	2
9960	2014-03-15	-3.55	1
9959	2014-03-14	+32.00	1

5. Finding the length of a series

```
| ID   | AMOUNT | SIGN | RN |
|------|--------|------|----|
| 9997 |  99.17 |    1 |  1 |
| 9981 |  71.44 |    1 |  2 |
| 9979 | -94.60 |   -1 |  3 |
| 9977 |  -6.96 |   -1 |  4 |
| 9971 | -65.95 |   -1 |  5 |
| 9964 |  15.13 |    1 |  6 |
| 9962 |  17.47 |    1 |  7 |
| 9960 |  -3.55 |   -1 |  8 |
| 9959 |  32.00 |    1 |  9 |
```

5. Finding the length of a series

That’s easy

```sql
SELECT
  id, amount,
  sign(amount) AS sign,
  row_number()
    OVER (ORDER BY id DESC) AS rn
FROM trx
```

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	
9977	-6.96	-1	4		
9971	-65.95	-1	5		5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	
9977	-6.96	-1	4		
9971	-65.95	-1	5		5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	
9977	-6.96	-1	4		
9971	-65.95	-1	5		5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	
9977	-6.96	-1	4		
9971	-65.95	-1	5		5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	
9977	-6.96	-1	4		
9971	-65.95	-1	5		5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	
9977	-6.96	-1	4		
9971	-65.95	-1	5		5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

LEAD() and LAG()

```
SELECT
  lag(v) OVER (ORDER BY v),
  v,
  lead(v) OVER (ORDER BY v)
FROM (
  VALUES (1), (2), (3), (4)
) t(v)
```

	lag integer	v integer	lead integer
1		1	2
2	1	2	3
3	2	3	4
4	3	4	

5. Finding the length of a series

```
SELECT
  trx.*,
  CASE WHEN lag(sign)
       OVER (ORDER BY id DESC) != sign
       THEN rn END AS lo,
  CASE WHEN lead(sign)
       OVER (ORDER BY id DESC) != sign
       THEN rn END AS hi,
FROM trx
```

5. Finding the length of a series

```
SELECT -- With NULL handling...
  trx.*,
  CASE WHEN coalesce(lag(sign)
            OVER (ORDER BY id DESC), 0)!=sign
            THEN rn END AS lo,
  CASE WHEN coalesce(lead(sign)
            OVER (ORDER BY id DESC), 0)!=sign
            THEN rn END AS hi,
FROM trx
```

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	2
9981	71.44	1	2	1	2
9979	-94.60	-1	3	3	5
9977	-6.96	-1	4	3	5
9971	-65.95	-1	5	3	5
9964	15.13	1	6	6	7
9962	17.47	1	7	6	7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	2
9981	71.44	1	2	1	2
9979	-94.60	-1	3	3	5
9977	-6.96	-1	4	3	5
9971	-65.95	-1	5	3	5
9964	15.13	1	6	6	7
9962	17.47	1	7	6	7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	2
9981	71.44	1	2	1	2
9979	-94.60	-1	3	3	5
9977	-6.96	-1	4	3	5
9971	-65.95	-1	5	3	5
9964	15.13	1	6	6	7
9962	17.47	1	7	6	7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	2
9981	71.44	1	2	1	2
9979	-94.60	-1	3	3	5
9977	-6.96	-1	4	3	5
9971	-65.95	-1	5	3	5
9964	15.13	1	6	6	7
9962	17.47	1	7	6	7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	2
9981	71.44	1	2	1	2
9979	-94.60	-1	3	3	5
9977	-6.96	-1	4	3	5
9971	-65.95	-1	5	3	5
9964	15.13	1	6	6	7
9962	17.47	1	7	6	7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	2
9981	71.44	1	2	1	2
9979	-94.60	-1	3	3	5
9977	-6.96	-1	4	3	5
9971	-65.95	-1	5	3	5
9964	15.13	1	6	6	7
9962	17.47	1	7	6	7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

```
SELECT
  trx.*,
  last_value (lo) IGNORE NULLS OVER (
    ORDER BY id DESC
    ROWS BETWEEN UNBOUNDED PRECEDING
    AND CURRENT ROW) AS lo,
  first_value(hi) IGNORE NULLS OVER (
    ORDER BY id DESC
    ROWS BETWEEN CURRENT ROW
    AND UNBOUNDED FOLLOWING) AS hi
FROM trx
```

5. Finding the length of a series

```
SELECT -- With NULL handling...
  trx.*,
  coalesce(last_value (lo) IGNORE NULLS OVER (
    ORDER BY id DESC
    ROWS BETWEEN UNBOUNDED PRECEDING
    AND CURRENT ROW), rn) AS lo,
  coalesce(first_value(hi) IGNORE NULLS OVER (
    ORDER BY id DESC
    ROWS BETWEEN CURRENT ROW
    AND UNBOUNDED FOLLOWING), rn) AS hi
FROM trx
```

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	
9977	-6.96	-1	4		
9971	-65.95	-1	5		5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI
9997	99.17	1	1	1	
9981	71.44	1	2		2
9979	-94.60	-1	3	3	5
9977	-6.96	-1	4	3	5
9971	-65.95	-1	5	3	5
9964	15.13	1	6	6	
9962	17.47	1	7		7
9960	-3.55	-1	8	8	8
9959	32.00	1	9	9	9

5. Finding the length of a series

Trivial last step

```
SELECT
  trx.*,
  1 + hi - lo AS length
FROM trx
```

5. Finding the length of a series

ID	AMOUNT	SIGN	RN	LO	HI	LENGTH
9997	99.17	1	1	1	2	2
9981	71.44	1	2	1	2	2
9979	-94.60	-1	3	3	5	3
9977	-6.96	-1	4	3	5	3
9971	-65.95	-1	5	3	5	3
9964	15.13	1	6	6	7	2
9962	17.47	1	7	6	7	2
9960	-3.55	-1	8	8	8	1
9959	32.00	1	9	9	9	1

5. Finding the length of a series

```
WITH
  trx1(id, amount, sign, rn) AS (
    SELECT id, amount, sign(amount), row_number() OVER (ORDER BY id DESC)
    FROM trx
  ),
  trx2(id, amount, sign, rn, lo, hi) AS (
    SELECT trx1.*,
    CASE WHEN coalesce(lag(sign) OVER (ORDER BY id DESC), 0) != sign
         THEN rn END,
    CASE WHEN coalesce(lead(sign) OVER (ORDER BY id DESC), 0) != sign
         THEN rn END
    FROM trx1
  )
SELECT
  trx2.*, 1
  - last_value (lo) IGNORE NULLS OVER (ORDER BY id DESC
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
  + first_value(hi) IGNORE NULLS OVER (ORDER BY id DESC
    ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
FROM trx2
```

Still OK?

6. The subset sum problem with SQL

What is the subset sum problem?

Explanation:

https://xkcd.com/287

(cannot include comic for © reasons. Please, don't use CC-BY SA NC without an actual commercial offering!)

Boring explanation:

https://en.wikipedia.org/wiki/Subset_sum_problem

6. The subset sum problem with SQL

```
For each of these...

| ID | TOTAL |
|----|-------|
|  1 | 25150 |
|  2 | 19800 |
|  3 | 27511 |
```

6. The subset sum problem with SQL

For each of these...

ID	TOTAL
1	25150
2	19800
3	27511

... find the closest sum from these...

ID	ITEM
1	7120
2	8150
3	8255
4	9051
5	1220
6	12515
7	13555
8	5221
9	812
10	6562

6. The subset sum problem with SQL

Desired result:

```
| TOTAL |  BEST | CALCULATION
|-------|-------|--------------------------------
| 25150 | 25133 | 7120 + 8150 + 9051 + 812
| 19800 | 19768 | 1220 + 12515 + 5221 + 812
| 27511 | 27488 | 8150 + 8255 + 9051 + 1220 + 812
```

6. The subset sum problem with SQL

Let's implement the naïvest possible, exponential algorithm

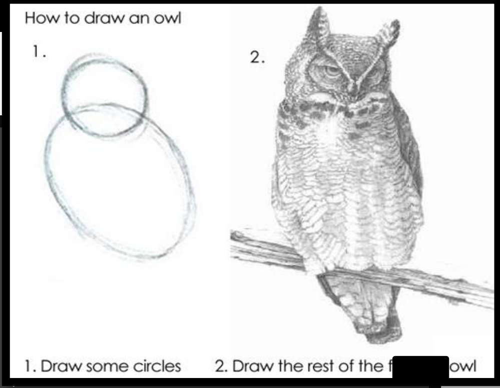
$$O(2^N N)$$

6. The subset sum problem with SQL

There are 2^N subsets and we need to sum at most N elements.

$$O(2^N N)$$

6. The subset sum problem with SQL




```
-- All the possible 2^N sums
WITH sums(sum, max_id, calc) AS (...)

-- Find the best sum per “TOTAL”
SELECT
  totals.total,
  something_something(total - sum) AS best,
  something_something(total - sum) AS calc
FROM draw_the_rest_of_the_*bleep*_owl
```

Maybe, if I just hide, the query will go away…?

Image credit: https://www.flickr.com/photos/12023825@N04/2898021822 By Peter. License CC-BY SA 2.0

What are the possible sums?

```
All the single-item sums      "Stack"

| ID   |  ITEM  |
|------|--------|
|    1 |  7120  |                SUMS(1:10)
|    2 |  8150  |
|    3 |  8255  |
|    4 |  9051  |
|    5 |  1220  |
|    6 | 12515  |
|    7 | 13555  |
|    8 |  5221  |
|    9 |   812  |
|   10 |  6562  |
```

6. The subset sum problem with SQL

What are the possible sums?

All the single-item sums   “Stack”

```
| ID   |  ITEM |
|------|-------|
```

ID	ITEM
~~1~~	~~7120~~
2	8150
3	8255
4	9051
5	1220
6	12515
7	13555
8	5221
9	812
10	6562

{ 7120 } x SUMS(2:10)

6. The subset sum problem with SQL

What are the possible sums?

All the single-item sums

```
| ID   |  ITEM |
|------|-------|
```

ID	ITEM
~~1~~	~~7120~~
~~2~~	~~8150~~
3	8255
4	9051
5	1220
6	12515
7	13555
8	5221
9	812
10	6562

"Stack"

{ 7120 + 8150 } x SUMS(3:10)

What are the possible sums?

All the single-item sums "Stack"

```
| ID   |  ITEM |
|------|-------|
|    1 |  7120 |
|    2 |  8150 |
|    3 |  8255 |
|    4 |  9051 |
|    5 |  1220 |
|    6 | 12515 |
|    7 | 13555 |
|    8 |  5221 |
|    9 |   812 |
|   10 |  6562 |
```

{ 7120 + 1220 } x SUMS(6:10)

6. The subset sum problem with SQL

What are the possible sums?

All the single-item sums   “Stack”

```
| ID    |   ITEM | IN_THE_SET |
|------|-------|------------|
```

ID	ITEM	IN_THE_SET
~~1~~	~~7120~~	1
~~2~~	~~8150~~	0
~~3~~	~~8255~~	0
~~4~~	~~9051~~	0
~~5~~	~~1220~~	1
6	12515	_
7	13555	_
8	5221	_
9	812	_
10	6562	_

{ 7120 + 1220 } x SUMS(6:10)

```sql
-- All the possible 2^N sums
WITH sums(sum, id, calc) AS (

  -- First iteration
  SELECT item, id, to_char(item)
  FROM items
)
```

```sql
-- All the possible 2^N sums
WITH sums(sum, id, calc) AS (

  -- First iteration
  SELECT item, id, to_char(item)
  FROM items

  -- Recursion
  UNION ALL
  SELECT
    item + sum,
    items.id,
    calc || ' + ' || item
  FROM sums JOIN items ON sums.id < items.id
)
```

```sql
-- All the possible 2^N sums
WITH sums(sum, id, calc) AS (

  -- First iteration
  SELECT item, id, to_char(item)
  FROM items

  -- Recursion
  UNION ALL
  SELECT
    item + sum,
    items.id,
    calc || ' + ' || item
  FROM sums JOIN items ON sums.id < items.id
)
```

```sql
-- All the possible 2^N sums
WITH sums(sum, id, calc) AS (

  -- First iteration
  SELECT item, id, to_char(item)
  FROM items

  -- Recursion
  UNION ALL
  SELECT
    item + sum,
    items.id,
    calc || ' + ' || item
  FROM sums JOIN items ON sums.id < items.id
)
```

```sql
-- All the possible 2^N sums
WITH sums(sum, id, calc) AS (

  -- First iteration
  SELECT item, id, to_char(item)
  FROM items

  -- Recursion
  UNION ALL
  SELECT
    item + sum,
    items.id,
    calc || ' + ' || item
  FROM sums JOIN items ON sums.id < items.id
)
```

```sql
-- All the possible 2^N sums
WITH sums(sum, id, calc) AS (

  -- First iteration
  SELECT item, id, to_char(item)
  FROM items

  -- Recursion
  UNION ALL
  SELECT
    item + sum,
    items.id,
    calc || ' + ' || item
  FROM sums JOIN items ON sums.id < items.id
)
```

6. The subset sum problem with SQL

For each of these...

ID	TOTAL
1	25150
2	19800
3	27511

... find the closest sum from these...

ID	ITEM
1	7120
2	8150
3	8255
4	9051
5	1220
6	12515
7	13555
8	5221
9	812
10	6562

6. The subset sum problem with SQL

```sql
-- All the possible 2^N sums
WITH sums(sum, max_id, calc) AS (...)

-- Find the best sum per “TOTAL”
SELECT
  totals.id,
  totals.total,
  min(abs(total - sum)) AS best_diff
FROM totals
CROSS JOIN sums
GROUP BY totals.id, totals.total
```

TOTAL	BEST_DIFF
25150	17
19800	32
27511	23

6. The subset sum problem with SQL

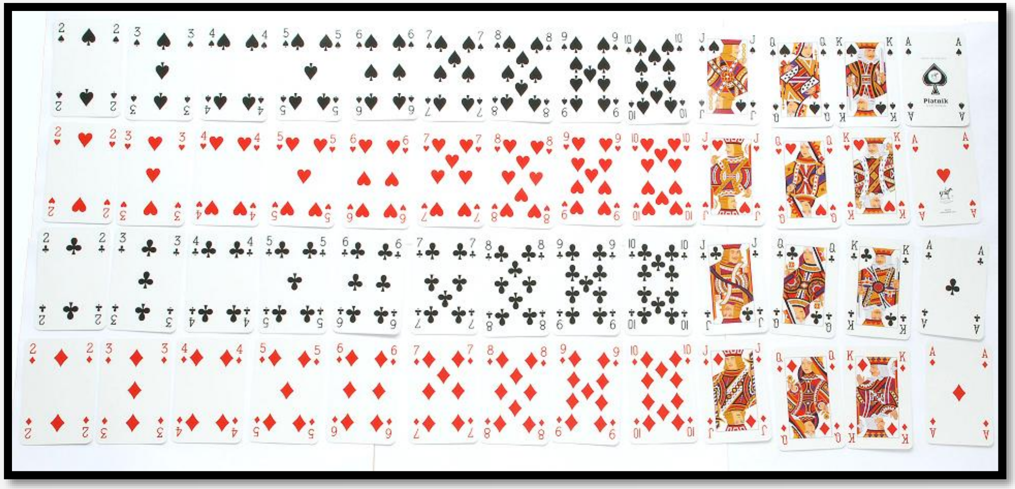
```
-- All the possible 2^N sums
WITH sums(sum, max_id, calc) AS (...)

-- Find the best sum per “TOTAL”
SELECT
  totals.id,
  totals.total,
  min(abs(total - sum)) AS best_diff
FROM totals
CROSS JOIN sums
GROUP BY totals.id, totals.total
```

TOTAL	BEST_DIFF
25150	17
19800	32
27511	23

What's this CROSS JOIN?

$$R \times S$$

Ranks = {A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2}
Suits = {♠, ♥, ♦, ♣}
Ranks × Suits = {
(A, ♠), (A, ♥), (A, ♦), (A, ♣),
(K, ♠), ...,
(2, ♠), (2, ♥), (2, ♦), (2, ♣)
}



By Trainler - Own work, CC BY 3.0,
https://commons.wikimedia.org/w/index.php?curid=7104281

6. The subset sum problem with SQL

```sql
SELECT
  totals.id,
  totals.total,
  min (sum) KEEP (
    DENSE_RANK FIRST ORDER BY abs(total - sum)
  ) AS best,
  min (calc) KEEP (
    DENSE_RANK FIRST ORDER BY abs(total - sum)
  ) AS calc,
FROM totals
CROSS JOIN sums
GROUP BY totals.id, totals.total
```

TOTAL	BEST	CALC
25150	25133	7120 + 8150 + 9051 + 812
19800	19768	1220 + 12515 + 5221 + 812
27511	27488	8150 + 8255 + 9051 + 1220 + 812

ORACLE

```sql
WITH sums(sum, id, calc) AS (
  SELECT item, id, to_char(item) FROM items
  UNION ALL
  SELECT item + sum, items.id, calc || ' + ' || item
  FROM sums JOIN items ON sums.id < items.id
)
SELECT
  totals.id,
  totals.total,
  min (sum) KEEP (
    DENSE_RANK FIRST ORDER BY abs(total - sum)
  ) AS best,
  min (calc) KEEP (
    DENSE_RANK FIRST ORDER BY abs(total - sum)
  ) AS calc,
FROM totals
CROSS JOIN sums
GROUP BY totals.id, totals.total
```

Electrified?

The running total must not be < 0

7. Capping a running total

DATE	AMOUNT
2012-01-01	800
2012-02-01	1900
2012-03-01	1750
2012-04-01	-20000
2012-05-01	900
2012-06-01	3900
2012-07-01	-2600
2012-08-01	-2600
2012-09-01	2100
2012-10-01	-2400
2012-11-01	1100
2012-12-01	1300

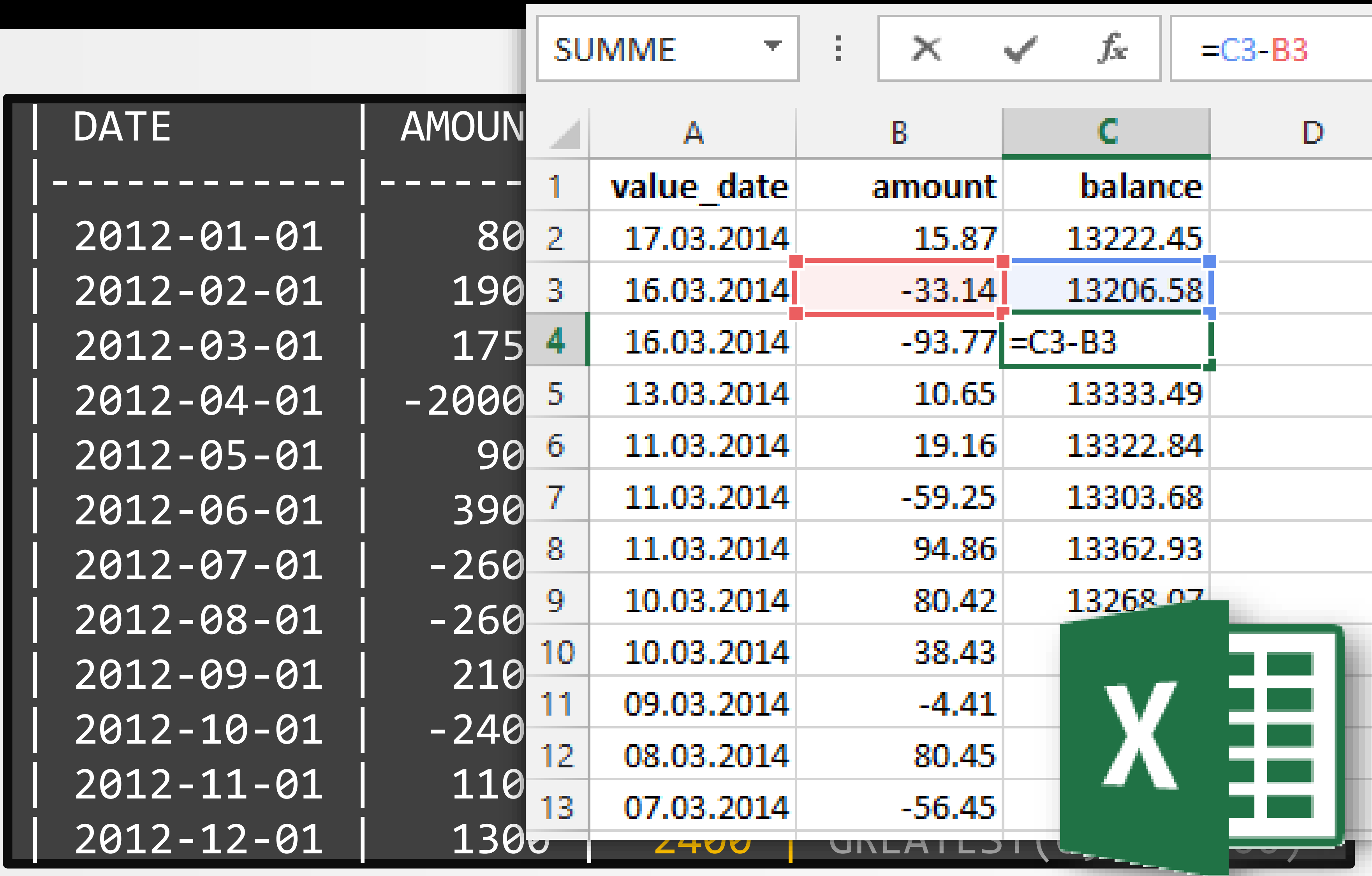
7. Capping a running total

DATE	AMOUNT	TOTAL
2012-01-01	800	800
2012-02-01	1900	2700
2012-03-01	1750	4450
2012-04-01	-20000	0
2012-05-01	900	900
2012-06-01	3900	4800
2012-07-01	-2600	2200
2012-08-01	-2600	0
2012-09-01	2100	2100
2012-10-01	-2400	0
2012-11-01	1100	1100
2012-12-01	1300	2400

7. Capping a running total

DATE	AMOUNT	TOTAL	
2012-01-01	800	800	GREATEST(0, 800)
2012-02-01	1900	2700	GREATEST(0, 2700)
2012-03-01	1750	4450	GREATEST(0, 4450)
2012-04-01	-20000	0	GREATEST(0, -15550)
2012-05-01	900	900	GREATEST(0, 900)
2012-06-01	3900	4800	GREATEST(0, 4800)
2012-07-01	-2600	2200	GREATEST(0, 2200)
2012-08-01	-2600	0	GREATEST(0, -400)
2012-09-01	2100	2100	GREATEST(0, 2100)
2012-10-01	-2400	0	GREATEST(0, -300)
2012-11-01	1100	1100	GREATEST(0, 1100)
2012-12-01	1300	2400	GREATEST(0, 2400)

7. Capping a running total



Reactive programming!

How to do it?

How to do it?

1. Window functions?

How to do it?

1. ~~Window functions?~~
   Probably not possible
2. Recursive SQL?

7. Capping a running total

How to do it?

1. ~~Window functions?~~
   Probably not possible
2. ~~Recursive SQL?~~
   Not geeky enough
3. Obscure, vendor-specific SQL?

How to do it?

1. ~~Window functions?~~
   Probably not possible
2. ~~Recursive SQL?~~
   Not geeky enough
3. Obscure, vendor-specific SQL?
   Jackpot.
   Someone else will maintain it.

7. Capping a running total

Oracle MODEL: Spreadsheet SQL!

```
SELECT ... FROM some_table

-- Put this after any table
MODEL ...
```

7. Capping a running total

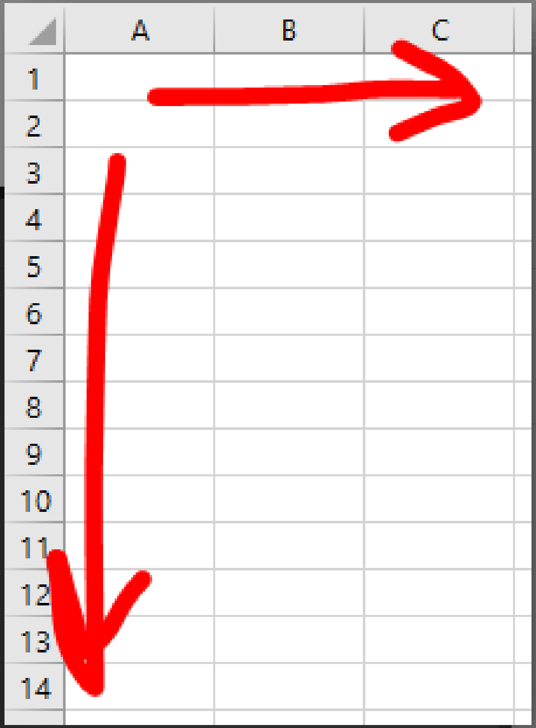
Oracle MODEL clause

```
MODEL
  -- The spreadsheet dimensions
  DIMENSION BY ...

  -- The spreadsheet cell type
  MEASURES ...

  -- The spreadsheet formulas
  RULES ...
```

7. Capping a running total

Oracle MODEL clause

```
MODEL
  -- The spreadsheet dimensions
  DIMENSION BY ...

  -- The spreadsheet cell type
  MEASURES ...

  -- The spreadsheet formulas
  RULES ...
```

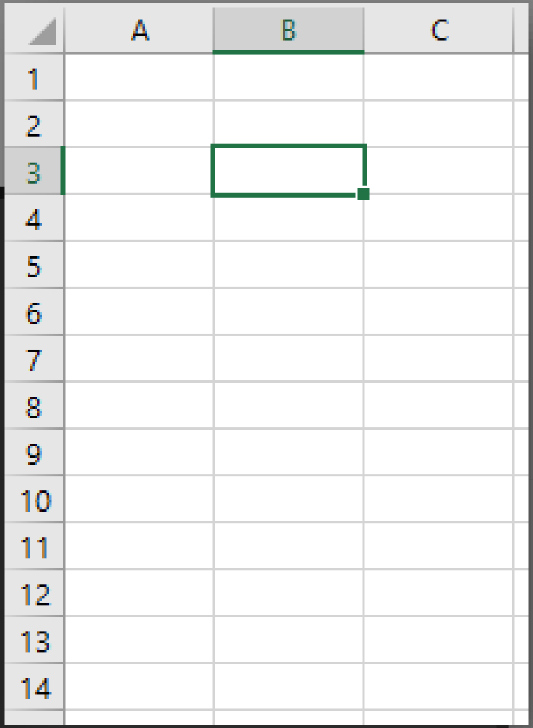


ORACLE

7. Capping a running total

Oracle MODEL clause

```
MODEL
  -- The spreadsheet dimensions
  DIMENSION BY ...

  -- The spreadsheet cell type
  MEASURES ...

  -- The spreadsheet formulas
  RULES ...
```

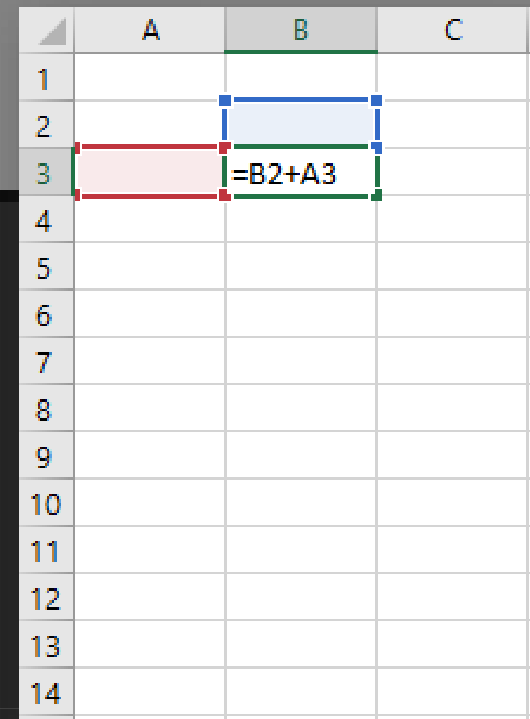


ORACLE

7. Capping a running total

Oracle MODEL clause

```
MODEL
  -- The spreadsheet dimensions
  DIMENSION BY ...

  -- The spreadsheet cell type
  MEASURES ...

  -- The spreadsheet formulas
  RULES ...
```



7. Capping a running total

```sql
SELECT *
FROM (
  SELECT date, amount, 0 AS total
  FROM amounts
)

-- Prepare the data
```

7. Capping a running total

```
SELECT *
FROM (
  SELECT date, amount, 0 AS total
  FROM amounts
)
MODEL
  DIMENSION BY (row_number() OVER (ORDER BY date) AS rn)

-- Individually enumerate each row with a row number
```

7. Capping a running total

```
SELECT *
FROM (
  SELECT date, amount, 0 AS total
  FROM amounts
)
MODEL
  DIMENSION BY (row_number() OVER (ORDER BY date) AS rn)
  MEASURES (date, amount, total)

-- Each «cell» contains these three values
```

7. Capping a running total

```
SELECT *
FROM (
  SELECT date, amount, 0 AS total
  FROM amounts
)
MODEL
  DIMENSION BY (row_number() OVER (ORDER BY date) AS rn)
  MEASURES (date, amount, total)
  RULES (
    total[any] = greatest(0,
      total[cv(rn) - 1] + amount[cv(rn)])
  )

-- «simple» rule based on cv(rn) (cv = current value)
```

7. Capping a running total

```
SELECT *
FROM (
  SELECT date, amount, 0 AS total
  FROM amounts
)
MODEL
  DIMENSION BY (row_number() OVER (ORDER BY date) AS rn)
  MEASURES (date, amount, total)
  RULES (
    total[any] = greatest(0, -- Getting NULLs right
      coalesce(total[cv(rn) - 1], 0) + amount[cv(rn)])
  )

-- «simple» rule based on cv(rn) (cv = current value)
```

7. Capping a running total

Read the whitepaper for more details:

http://www.oracle.com/technetwork/middleware/bi-foundation/10gr1-twp-bi-dw-sqlmodel-131067.pdf

(Google «Oracle MODEL Whitepaper»)

7. Capping a running total

Extra credit:

After this talk, do tricks #2 - #6 with MODEL!

(☞ﾟヮﾟ)☞ ☜(ﾟヮﾟ☜)

8. Time series pattern recognition

ID	VALUE_DATE	AMOUNT	LEN
9997	2014-03-18	+ 99.17	1
9981	2014-03-16	- 71.44	4
9979	2014-03-16	- 94.60	4
9977	2014-03-16	- 6.96	4
9971	2014-03-15	- 65.95	4
9964	2014-03-15	+ 15.13	3
9962	2014-03-15	+ 17.47	3
9960	2014-03-15	+ 3.55	3
9959	2014-03-14	- 32.00	1

8. Time series pattern recognition

ID	VALUE_DATE	AMOUNT	LEN	TRIGGER
9997	2014-03-18	+ 99.17	1	
9981	2014-03-16	- 71.44	4	
9979	2014-03-16	- 94.60	4	x
9977	2014-03-16	- 6.96	4	
9971	2014-03-15	- 65.95	4	
9964	2014-03-15	+ 15.13	3	
9962	2014-03-15	+ 17.47	3	
9960	2014-03-15	+ 3.55	3	
9959	2014-03-14	- 32.00	1	

8. Time series pattern recognition

ID	VALUE_DATE	AMOUNT	LEN	TRIGGER
9997	2014-03-18	+ 99.17	1	
9981	2014-03-16	- 71.44	4	
9979	2014-03-16	- 94.60	4	x
9977	2014-03-16	- 6.96	4	
9971	2014-03-15	- 65.95	4	
9964	2014-03-15	+ 15.13	3	
9962	2014-03-15	+ 17.47	3	
9960	2014-03-15	+ 3.55	3	
9959	2014-03-14	- 32.00	1	

8. Time series pattern recognition

Trigger on the 3rd repetition of an event if the event occurs *more* than 3 times.

How to do it?

1. ~~Window functions?~~
   Probably not possible
2. ~~Recursive SQL?~~
   Not geeky enough
3. Obscure, vendor-specific SQL?
   Jackpot.
   Someone else will maintain it.

8. Time series pattern recognition

Oracle 12c MATCH_RECOGNIZE!

```
SELECT ... FROM some_table

-- Put this after any table to pattern-match
-- the table's contents
MATCH_RECOGNIZE (...)
```

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY ...

  -- Pattern matching is done in this order
)
```

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY ...
  MEASURES ...

  -- These are the columns produced by matches
)
```

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY ...
  MEASURES ...
  ALL ROWS PER MATCH

  -- A short specification of what rows are
  -- returned from each match
)
```

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY ...
  MEASURES ...
  ALL ROWS PER MATCH
  PATTERN (...)

  -- «Regular expressions» of events to match
)
```

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY ...
  MEASURES ...
  ALL ROWS PER MATCH
  PATTERN (...)
  DEFINE ...

  -- The definitions of «what is an event»
)
```

ORACLE

Ready?

8. Time series pattern recognition

ID	VALUE_DATE	AMOUNT	TRIGGER
9997	2014-03-18	+ 99.17	
9981	2014-03-16	- 71.44	
9979	2014-03-16	- 94.60	x
9977	2014-03-16	- 6.96	
9971	2014-03-15	- 65.95	
9964	2014-03-15	+ 15.13	
9962	2014-03-15	+ 17.47	
9960	2014-03-15	+ 3.55	
9959	2014-03-14	- 32.00	

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY id
  MEASURES ...
  ALL ROWS PER MATCH
  PATTERN (...)
  DEFINE ...
)
```

ID	VALUE_DATE	AMOUNT	TRIGGER
9997	2014-03-18	+ 99.17	
9981	2014-03-16	- 71.44	
9979	2014-03-16	- 94.60	x
9977	2014-03-16	- 6.96	
9971	2014-03-15	- 65.95	
9964	2014-03-15	+ 15.13	
9962	2014-03-15	+ 17.47	
9960	2014-03-15	+ 3.55	
9959	2014-03-14	- 32.00	

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY id
  MEASURES classifier() AS trg
  ALL ROWS PER MATCH
  PATTERN (...)
  DEFINE ...
)
```

ID	VALUE_DATE	AMOUNT	TRIGGER
9997	2014-03-18	+ 99.17	
9981	2014-03-16	- 71.44	
9979	2014-03-16	- 94.60	x
9977	2014-03-16	- 6.96	
9971	2014-03-15	- 65.95	
9964	2014-03-15	+ 15.13	
9962	2014-03-15	+ 17.47	
9960	2014-03-15	+ 3.55	
9959	2014-03-14	- 32.00	

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY id
  MEASURES classifier() AS trg
  ALL ROWS PER MATCH
  PATTERN (S (R X R+)?)
  DEFINE ...
)
```

ID	VALUE_DATE	AMOUNT	TRIGGER
9997	2014-03-18	+ 99.17	
9981	2014-03-16	- 71.44	
9979	2014-03-16	- 94.60	x
9977	2014-03-16	- 6.96	
9971	2014-03-15	- 65.95	
9964	2014-03-15	+ 15.13	
9962	2014-03-15	+ 17.47	
9960	2014-03-15	+ 3.55	
9959	2014-03-14	- 32.00	

ORACLE

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY id
  MEASURES classifier() AS trg
  ALL ROWS PER MATCH
  PATTERN (S (R X R+)?)
  DEFINE
    R AS sign(R.amount) = prev(sign(R.amount)),
    X AS sign(X.amount) = prev(sign(X.amount))
)
```

ID	VALUE_DATE	AMOUNT	TRIGGER
9997	2014-03-18	+ 99.17	
9981	2014-03-16	- 71.44	
9979	2014-03-16	- 94.60	x
9977	2014-03-16	- 6.96	
9971	2014-03-15	- 65.95	
9964	2014-03-15	+ 15.13	
9962	2014-03-15	+ 17.47	
9960	2014-03-15	+ 3.55	
9959	2014-03-14	- 32.00	

8. Time series pattern recognition

```
SELECT *
FROM series
MATCH_RECOGNIZE (
  ORDER BY id
  MEASURES classifier() AS trg
  ALL ROWS PER MATCH
  PATTERN (S (R X R+)?)
  DEFINE
    R AS sign(R.amount) = prev(sign(R.amount)),
    X AS sign(X.amount) = prev(sign(X.amount))
)
```

ID	VALUE_DATE	AMOUNT	TRIGGER
9997	2014-03-18	+ 99.17	
9981	2014-03-16	- 71.44	
9979	2014-03-16	- 94.60	x
9977	2014-03-16	- 6.96	
9971	2014-03-15	- 65.95	
9964	2014-03-15	+ 15.13	
9962	2014-03-15	+ 17.47	
9960	2014-03-15	+ 3.55	
9959	2014-03-14	- 32.00	

8. Time series pattern recognition

```
PATTERN (S (R X R+)?)
```

```
|   ID | VALUE_DATE |  AM
|------|------------|----------|-----|
| 9997 | 2014-03-18 | + 99.17  |   S |
| 9981 | 2014-03-16 | - 71.44  |   R |
| 9979 | 2014-03-16 | - 94.60  |   X |
| 9977 | 2014-03-16 | -  6.96  |   R |
| 9971 | 2014-03-15 | - 65.95  |   S |
| 9964 | 2014-03-15 | + 15.13  |   S |
| 9962 | 2014-03-15 | + 17.47  |   S |
| 9960 | 2014-03-15 | +  3.55  |   S |
| 9959 | 2014-03-14 | - 32.00  |   S |
```

8. Time series pattern recognition

```
PATTERN (S (R X R+)?)
```

ID	VALUE_DATE	AM	
9997	2014-03-18	+ 99.17	S
9981	2014-03-16	- 71.44	R
9979	2014-03-16	- 94.60	X
9977	2014-03-16	- 6.96	R
9971	2014-03-15	- 65.95	S
9964	2014-03-15	+ 15.13	S
9962	2014-03-15	+ 17.47	S
9960	2014-03-15	+ 3.55	S
9959	2014-03-14	- 32.00	S

8. Time series pattern recognition

```
PATTERN (S (R X R+)?)
```

ID	VALUE_DATE	AM	
9997	2014-03-18	+ 99.17	S
9981	2014-03-16	- 71.44	R
9979	2014-03-16	- 94.60	X
9977	2014-03-16	- 6.96	R
9971	2014-03-15	- 65.95	S
9964	2014-03-15	+ 15.13	S
9962	2014-03-15	+ 17.47	S
9960	2014-03-15	+ 3.55	S
9959	2014-03-14	- 32.00	S

8. Time series pattern recognition

```
PATTERN (S (R X R+)?)
```

ID	VALUE_DATE	AM	
9997	2014-03-18	+ 99.17	S
9981	2014-03-16	- 71.44	R
9979	2014-03-16	- 94.60	X
9977	2014-03-16	- 6.96	R
9971	2014-03-15	- 65.95	S
9964	2014-03-15	+ 15.13	S
9962	2014-03-15	+ 17.47	S
9960	2014-03-15	+ 3.55	S
9959	2014-03-14	- 32.00	S

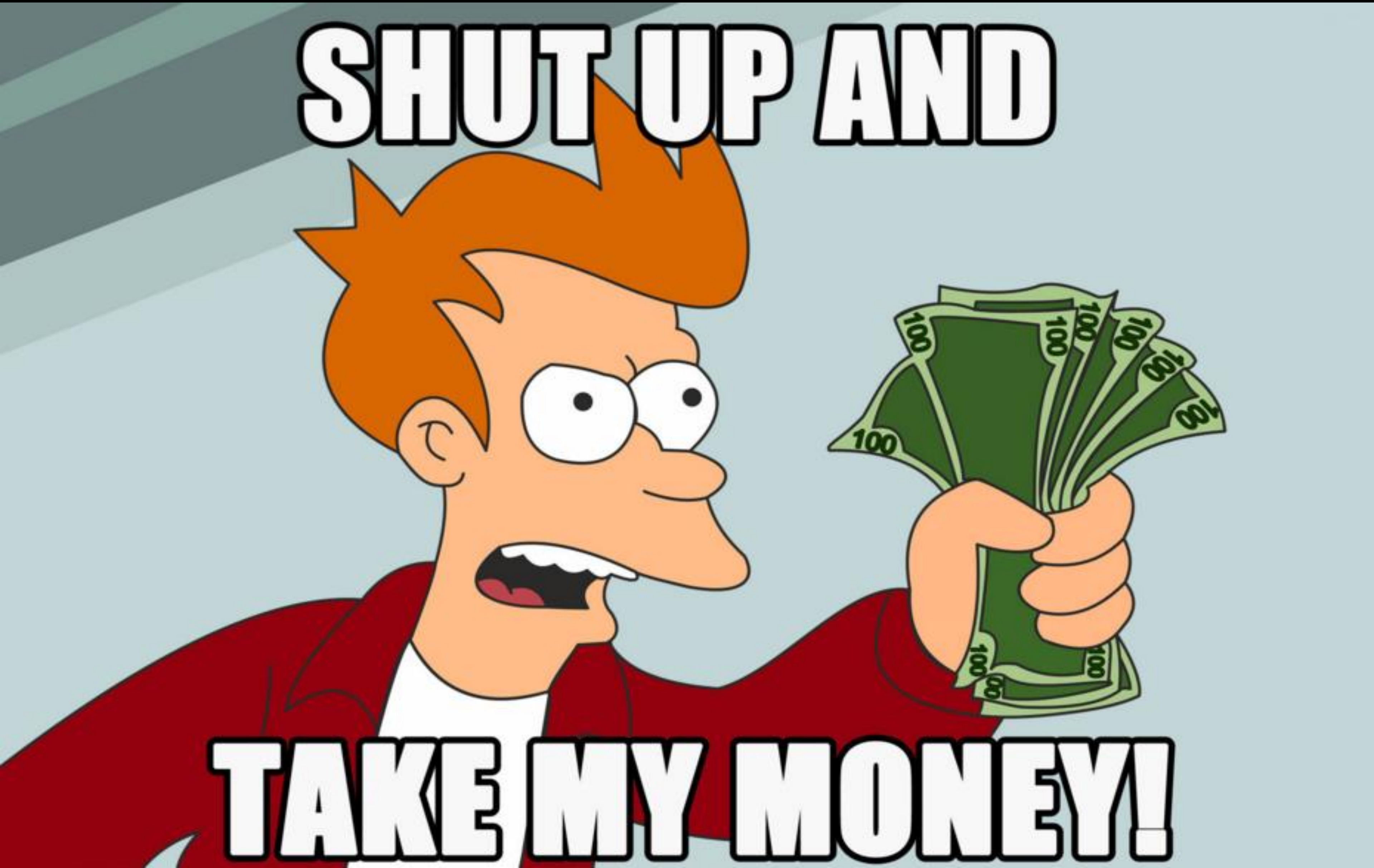
8. Time series pattern recognition

```
SELECT
  id, value_date, amount,
  CASE trg WHEN 'X' THEN 'X' END trg
FROM series
MATCH_RECOGNIZE (
  ORDER BY id
  MEASURES classifier() AS trg
  ALL ROWS PER MATCH
  PATTERN (S (R X R+)?)
  DEFINE
    R AS sign(R.amount) = prev(sign(R.amount)),
    X AS sign(X.amount) = prev(sign(X.amount))
)
```

8. Time series pattern recognition

ID	VALUE_DATE	AMOUNT	TRG
9997	2014-03-18	+ 99.17	
9981	2014-03-16	- 71.44	
9979	2014-03-16	- 94.60	X
9977	2014-03-16	- 6.96	
9971	2014-03-15	- 65.95	
9964	2014-03-15	+ 15.13	
9962	2014-03-15	+ 17.47	
9960	2014-03-15	+ 3.55	
9959	2014-03-14	- 32.00	

8. Time series pattern recognition



Read the whitepaper for more details:

http://www.oracle.com/ocom/groups/public/@otn/documents/webcontent/1965433.pdf

(Google «Oracle MATCH_RECOGNIZE Whitepaper»)

Extra credit:

After this talk, do tricks #2 - #7 with MATCH_RECOGNIZE!

☜(˚▽˚)☞

Now that you're experts…

… this is almost too embarassingly simple

9. Pivoting and unpivoting

NAME	TITLE	RATING
A. GRANT	ANNIE IDENTITY	G
A. GRANT	DISCIPLE MOTHER	PG
A. GRANT	GLORY TRACY	PG-13
A. HUDSON	LEGEND JEDI	PG
A. CRONYN	IRON MOON	PG
A. CRONYN	LADY STAGE	PG
B. WALKEN	SIEGE MADRE	R

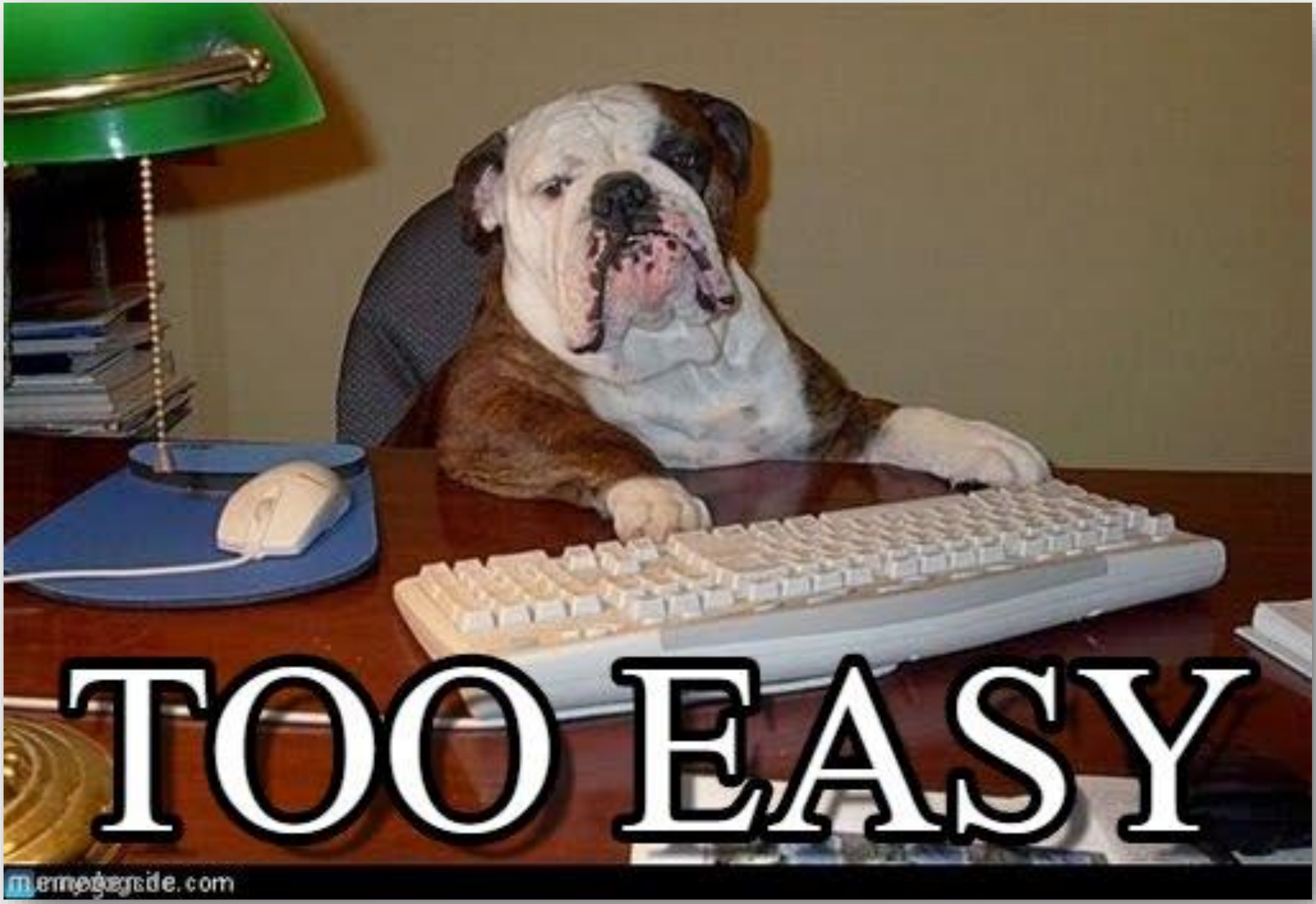
Pivoting

NAME	NC-17	PG	G	PG-13	R
A. GRANT	3	6	5	3	1
A. HUDSON	12	4	7	9	2
A. CRONYN	6	9	2	6	4
B. WALKEN	8	8	4	7	3
B. WILLIS	5	5	14	3	6
C. DENCH	6	4	5	4	5
C. NEESON	3	8	4	7	3

Unpivoting

NAME	RATING	COUNT
A. GRANT	NC-17	3
A. GRANT	PG	6
A. GRANT	G	5
A. GRANT	PG-13	3
A. GRANT	R	6
A. HUDSON	NC-17	12
A. HUDSON	PG	4

OK? – I know. That meme again…



Only PostgreSQL so far

```
SELECT
  first_name, last_name,
  count(*) FILTER (WHERE rating = 'NC-17') AS "NC-17",
  count(*) FILTER (WHERE rating = 'PG'   ) AS "PG",
  count(*) FILTER (WHERE rating = 'G'    ) AS "G",
  count(*) FILTER (WHERE rating = 'PG-13') AS "PG-13",
  count(*) FILTER (WHERE rating = 'R'    ) AS "R"
FROM actor AS a
JOIN film_actor AS fa USING (actor_id)
JOIN film AS f USING (film_id)
GROUP BY actor_id
```

All others

```
SELECT
  first_name, last_name,
  count(CASE rating WHEN 'NC-17' THEN 1 END) AS "NC-17",
  count(CASE rating WHEN 'PG'    THEN 1 END) AS "PG",
  count(CASE rating WHEN 'G'     THEN 1 END) AS "G",
  count(CASE rating WHEN 'PG-13' THEN 1 END) AS "PG-13",
  count(CASE rating WHEN 'R'     THEN 1 END) AS "R"
FROM actor AS a
JOIN film_actor AS fa USING (actor_id)
JOIN film AS f USING (film_id)
GROUP BY actor_id
```

9. Pivoting and unpivoting

```
SELECT
  actor_id, first_name, last_name,
  "NC-17", "PG", "G", "PG-13", "R"
FROM (
  SELECT actor_id, first_name, last_name, rating
  FROM actor a
  JOIN film_actor fa USING (actor_id)
  JOIN film f USING (film_id)
)
PIVOT (
  count(*) FOR rating IN (
    'NC-17' AS "NC-17",
    'PG'    AS "PG",
    'G'     AS "G",
    'PG-13' AS "PG-13",
    'R'     AS "R"
  )
)
```

9. Pivoting and unpivoting

```
SELECT something, something
FROM some_table
PIVOT (
  count(*) FOR rating IN (
    'NC-17' AS "NC-17",
    'PG'    AS "PG",
    'G'     AS "G",
    'PG-13' AS "PG-13",
    'R'     AS "R"
  )
)
```

9. Pivoting and unpivoting

```
SELECT something, something
FROM some_table
UNPIVOT (
  count    FOR rating IN (
    "NC-17" AS 'NC-17',
    "PG"    AS 'PG',
    "G"     AS 'G',
    "PG-13" AS 'PG-13',
    "R"     AS 'R'
  )
)
```

9. Pivoting and unpivoting

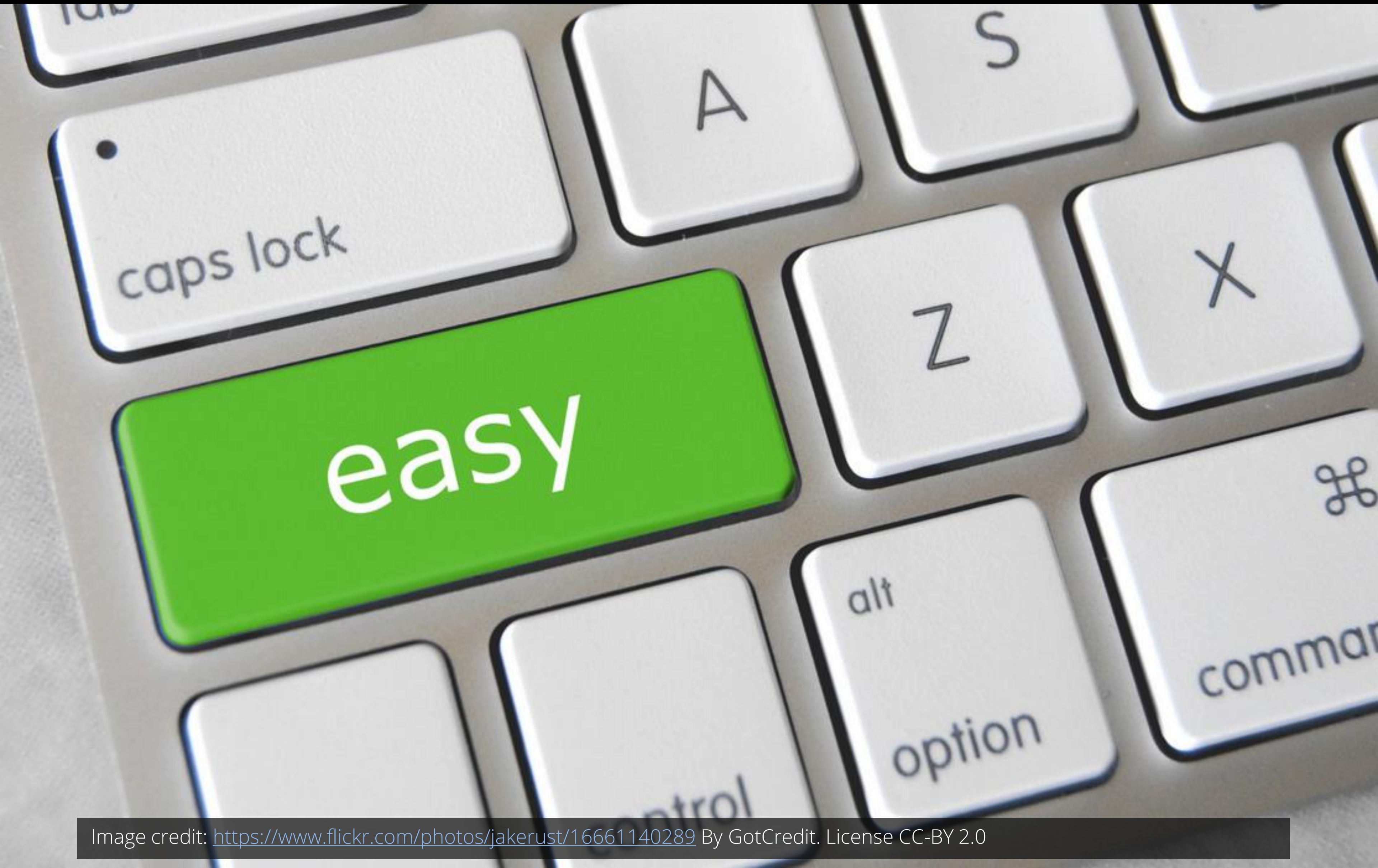
Pivoting:

Values from a single column become columns containing aggregations

Unpivoting:

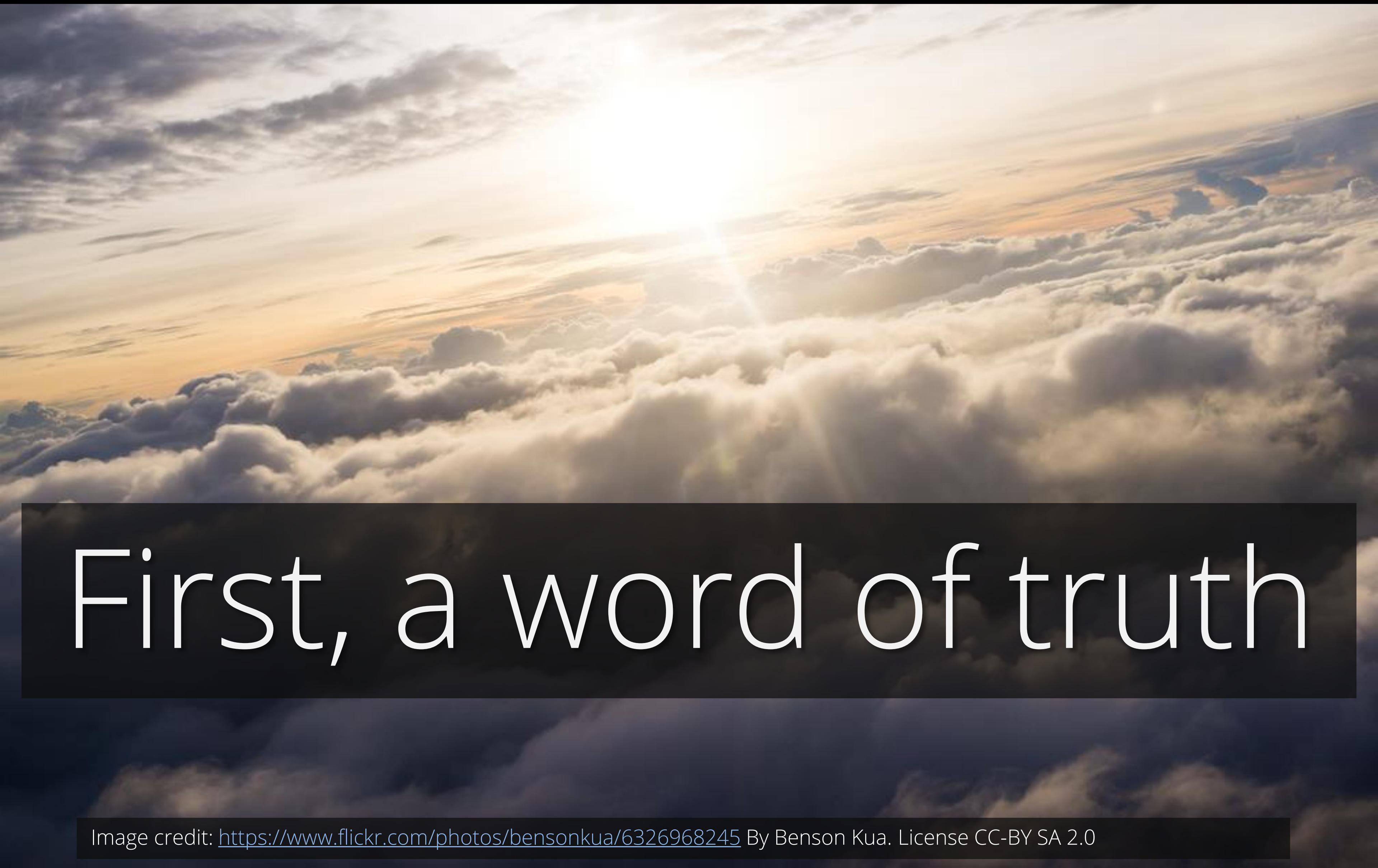
Columns become values in a single column

9. Pivoting and unpivoting



XML and JSON in the database

10. Abusing XML and JSON



JSON is just XML with less features and less syntax

10. Abusing XML and JSON

Everyone knows:

XML is awesome.

10. Abusing XML ~~and JSON~~

Corollary:

JSON is less awesome

Side note

XSLT is the only thing even more awesome than SQL

10. Abusing XML ~~and JSON~~

```
<actors>
  <actor>
    <first-name>Bud</first-name>
    <last-name>Spencer</last-name>
    <films>God Forgives... I Don’t, Double Trouble, They Call Him Bulldozer</films>
  </actor>
  <actor>
    <first-name>Terence</first-name>
    <last-name>Hill</last-name>
    <films>God Forgives... I Don’t, Double Trouble, Lucky Luke</films>
  </actor>
</actors>
```

actor_id bigint	first_name text	last_name text	film_id integer	film text
1	Bud	Spencer	1	God Forgives... I Don't
2	Terence	Hill	1	God Forgives... I Don't
1	Bud	Spencer	2	Double Trouble
2	Terence	Hill	2	Double Trouble
1	Bud	Spencer	3	They Call Him Bulldozer
2	Terence	Hill	3	Lucky Luke

10. Abusing XML ~~and JSON~~

```
WITH RECURSIVE
  x(v) AS (SELECT '...'::xml),
  actors(
    actor_id, first_name, last_name, films
  ) AS (...),
  films(
    actor_id, first_name, last_name,
    film_id, film
  ) AS (...)
SELECT *
FROM films
```

10. Abusing XML ~~and JSON~~

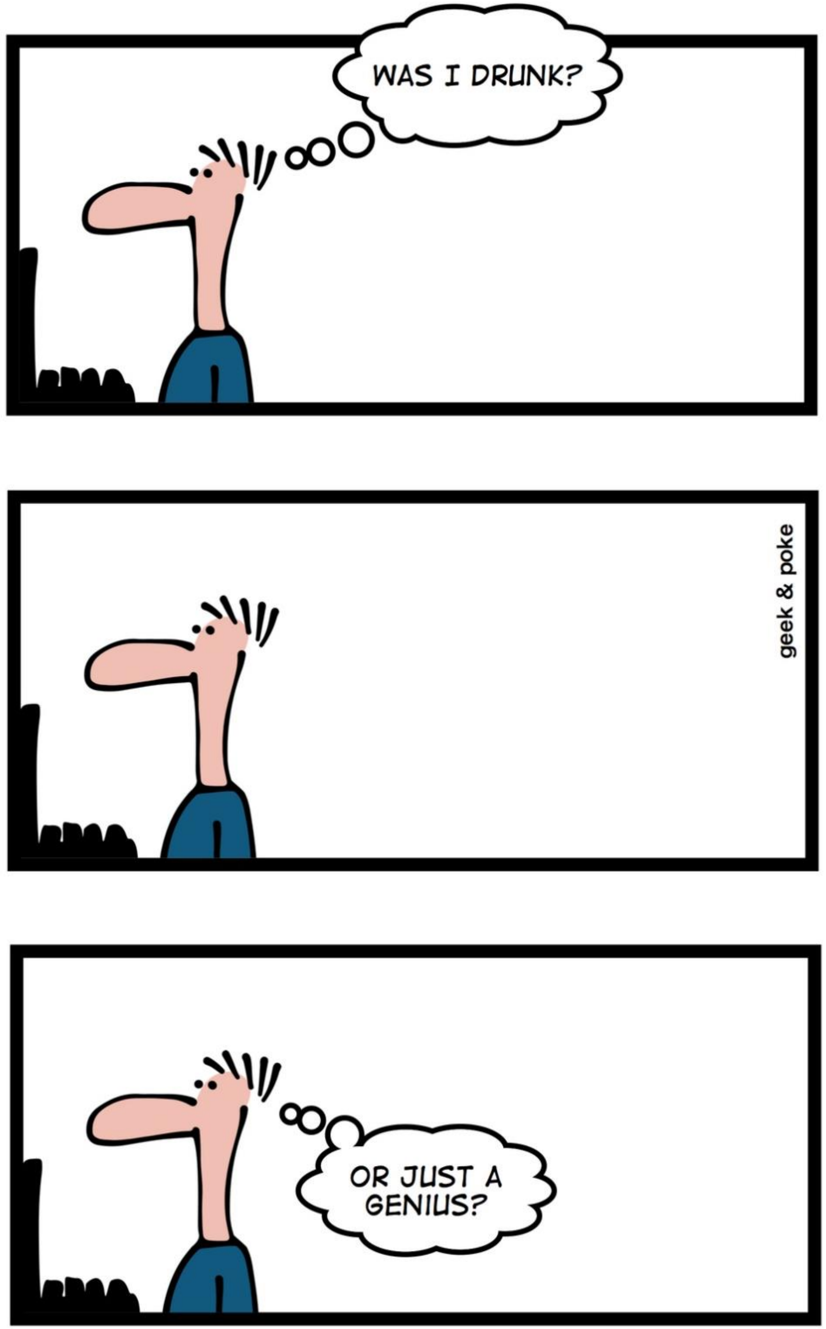
```
WITH RECURSIVE
  x(v) AS (SELECT '
<actors>
  <actor>
    <first-name>Bud</first-name>
    <last-name>Spencer</last-name>
    <films>God Forgives... I Don’t, Double Trouble, They Call Him
Bulldozer</films>
  </actor>
  <actor>
    <first-name>Terence</first-name>
    <last-name>Hill</last-name>
    <films>God Forgives... I Don’t, Double Trouble, Lucky Luke</films>
  </actor>
</actors>'::xml),
  actors(actor_id, first_name, last_name, films) AS (...),
  films(actor_id, first_name, last_name, film_id, film) AS (...)
SELECT *
FROM films
```

10. Abusing XML ~~and JSON~~

```
WITH RECURSIVE
  x(v) AS (SELECT '...'::xml),
  actors(actor_id, first_name, last_name, films) AS (
    SELECT
      row_number() OVER (),
      (xpath('//first-name/text()', t.v))[1]::TEXT,
      (xpath('//last-name/text()' , t.v))[1]::TEXT,
      (xpath('//films/text()'     , t.v))[1]::TEXT
    FROM unnest(xpath('//actor', (SELECT v FROM x))) t(v)
  ),
  films(actor_id, first_name, last_name, film_id, film)
AS (...)
SELECT *
FROM films
```

10. Abusing XML ~~and JSON~~

```
WITH RECURSIVE
  x(v) AS (SELECT '...'::xml),
  actors(actor_id, first_name, last_name, films) AS (...),
  films(actor_id, first_name, last_name, film_id, film) AS (
    SELECT actor_id, first_name, last_name, 1,
      regexp_replace(films, ',.+', '')
    FROM actors
    UNION ALL
    SELECT actor_id, a.first_name, a.last_name, f.film_id + 1,
      regexp_replace(a.films, '.*' || f.film || ', ?(.*?)(,.+)?', '\1')
    FROM films AS f
    JOIN actors AS a USING (actor_id)
    WHERE a.films NOT LIKE '%' || f.film
  )
SELECT *
FROM films
```



YESTERDAYS REGEX

10 SQL tricks to convince you SQL is awesome

1. Everything is a table
2. Data generation with recursive SQL
3. Running total calculations
4. Finding the length of a series
5. Finding the largest series with no gaps
6. The subset sum problem with SQL
7. Capping a running total
8. Time series pattern recognition
9. Pivoting and unpivoting
10. Abusing XML and JSON (don't do this at home)

Noun

awe (uncountable)

1. A feeling of fear and reverence.
2. A feeling of amazement.

Noun

awe (uncountable)

1. A feeling of fear and reverence.
2. A feeling of amazement.

Noun

maze (plural mazes)

1. A labyrinth; a puzzle consisting of a complicated network of paths or passages, the aim of which is to find one's way.
2. Something made up of many confused or conflicting elements; a tangle.
3. Confusion of thought; perplexity; uncertainty; state of bewilderment.

Noun

maze (plural mazes)

1. A labyrinth; a puzzle consisting of a complicated network of paths or passages, the aim of which is to find one's way.
2. Something made up of many confused or conflicting elements; a tangle.
3. Confusion of thought; perplexity; uncertainty; state of bewilderment.

Why do I talk about SQL?

SQL is the only ever successful, mainstream, and general-purpose 4GL (Fourth-Generation Programming Language)

And it is awesome!

Why doesn’t anyone else talk about SQL?

¯_(シ)_/¯

Can I write SQL in Java?

```
Result<Record2<String, String>> result
dsl().select(
        ACTOR.FIRST_NAME,
        ACTOR.LAST_NAME)
    .from(ACTOR)
    .join(FILM_ACTOR)
    .on(ACTOR.ACTOR_ID.eq(FILM_ACTOR.ACTOR_ID))
    .where(ACTOR.FIRST_NAME.like("A%"))
    .fetch();
```

So what's the key takeaway?

1. Can you do it in the database?

So what's the key takeaway?

1. Can you do it in the database? Yes

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database?

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *<u>you</u>* do it in the database? Yes

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *<u>you</u>* do it in the database? Yes
   (… after visiting my 2 day SQL training)

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
   (in Zurich, with 42talents.com!
   http://42talents.com/de/training/technical/SQL-Masterclass/)

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (... after visiting my 2 day SQL training)
3. Can you do it in *your* database?

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes
   (… unless you're using MySQL)

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *<u>you</u>* do it in the database? Yes (… after visiting my 2 day SQL training)
3. Can you do it in *<u>your</u>* database? Yes (… unless you're using MySQL)
4. *<u>Should</u>* you do it in the database?

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes
   (… unless you're using MySQL)
4. *Should* you do it in the database? No

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes (... after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes (... unless you're using MySQL)
4. *Should* you do it in the database? No

JUST KIDDING!

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (... after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes
   (... unless you're using MySQL)
4. *Should* you do it in the database? Yes

So what’s the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes
   (… unless you're using MySQL)
4. *Should* you do it in the database? Yes
5. Do listicles attract attention?

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes
   (… unless you're using MySQL)
4. *Should* you do it in the database? Yes
5. Do listicles attract attention? Yes

So what's the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes
   (… unless you're using MySQL)
4. *Should* you do it in the database? Yes
5. Do listicles attract attention? Yes
6. Will this talk ever end?

So what’s the key takeaway?

1. Can you do it in the database? Yes
2. Can *you* do it in the database? Yes
   (… after visiting my 2 day SQL training)
3. Can you do it in *your* database? Yes
   (… unless you're using MySQL)
4. *Should* you do it in the database? Yes
5. Do listicles attract attention? Yes
6. Will this talk ever end? Yes

Thank you

Contact me if you want this talk as a 1 or 2 day in-house training about awesome SQL!

Our partner in Zurich:
http://42talents.com/de/training/technical/SQL-Masterclass/

Coordinates

- Blog: http://blog.jooq.org (excellent Java SQL content)
- Twitter: @JavaOOQ / @lukaseder (more lame jokes)
- E-Mail: lukas.eder@datageekery.com
- Bank account: CH57 8148 7000 0SQL AWSM 7