Czemu SQL?

Praktyka w pracy z (dużymi) danymi

Jakub Nowacki
Confitura 2017

O mnie

- Big Data Scientist @ SigDelta
- Trainer @ Sages
- #Spark
- #Scala
- #Python
- Twitter: @jsnowacki
- GitHub: https://github.com/jsnowacki/
- LinkedIn: https://www.linkedin.com/in/jakubnowacki

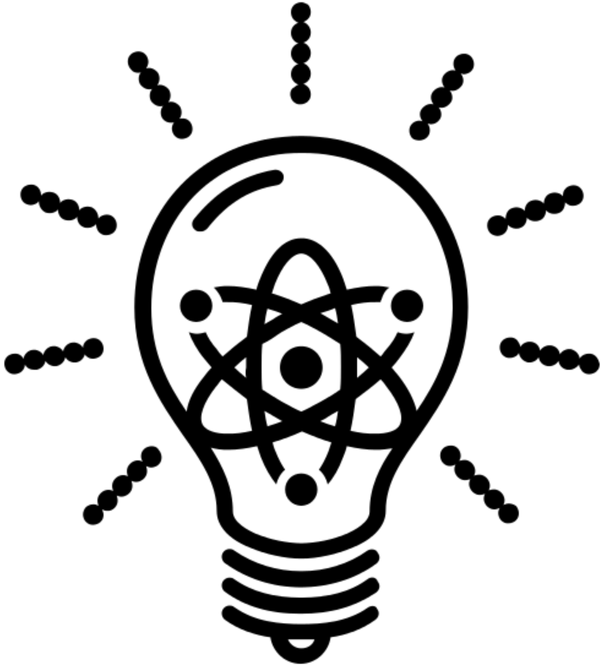
O czym ta prezentacja nie jest?

- O bazach danych (choć się pojawią)
- O ORM (ani pozytywnie, ani negatywnie)
- O jednej technologii (ale technologie będą)
- O mikroserwisach (choć może się coś komuś przyda)
- O Big Data (nie wszystko Big, co się świeci)
- Pieśnią pochwalną SQL (bo nie lubię go aż tak)

Co będzie?

- Co to SQL?
- Dlaczego SQL jest popularny?
- Dlaczego jest użyteczny (na przykładach)?
- Co zainspirował?
- Jak możemy używać?
- Do czego się przyda?

Perspektywa dano-centryczna



Czym jest SQL?

- Jest popularny i szeroko używany
  - Bazy relacyjne
  - Hurtownie danych
  - Narzędzia Business Intelligence
  - Systemy Big Data (Spark, Hive, Presto, Druid, Google BigQuery, AWS Athena itp.)
- Jest dostosowany do pracy z danymi
- Jest językiem deklaratywnym, więc mówi o tym co, a nie jak
- Jest intensywnie rozwijany
  - Specyfikacje od 1986 (SQL-86) do 2016 (SQL:2016)

Składnia SQL

- Data Definition Language (DDL)
  - CREATE
  - DROP
  - ALTER
- Data Manipulation Language (DML)
  - INSERT
  - UPDATE
  - DELETE
- Data Control Language (DCL)
  - GRANT
  - REVOKE
  - DENY
- Data Query Language (DQL)
  - SELECT

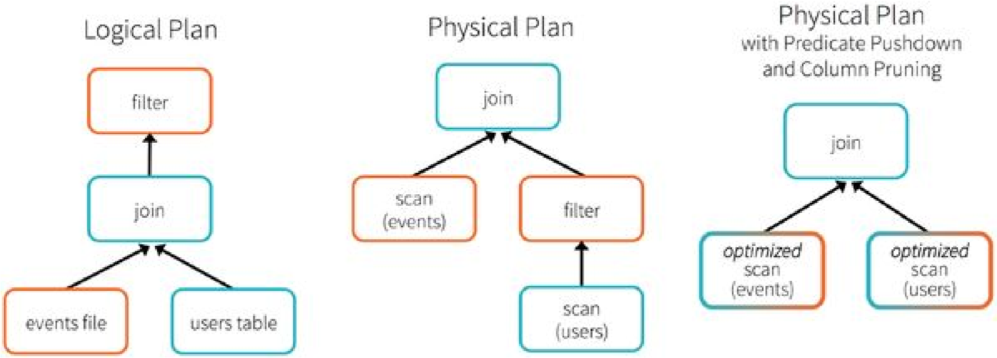
Model danych

- Forma tabeli, zatem ustrukturyzowane dane
    - Wyróżniamy zarówno wiersze jak i kolumny
    - Wiele systemów wspiera struktury w kolumnach
- System (w miarę) jednolitych typów, np: INTEGER, BOOLEAN, TEXT, TIMESTAMP itp.
- Dość prosty do zrozumienia, także dla osób nietechnicznych

```
CREATE TABLE example (
     id INTEGER,
     user VARCHAR(100),
     bio TEXT
);
```

Deklaratywny (!)

- Definiujemy co, a nie jak, co poprawia przenośność
- Platforma zajmuje się optymalizacją
- Ograniczony zestaw operacji pozwala na lepszą optymalizację, bo silnik wie co możemy zrobić
- W gruncie rzeczy, często powtarzamy te same czynności

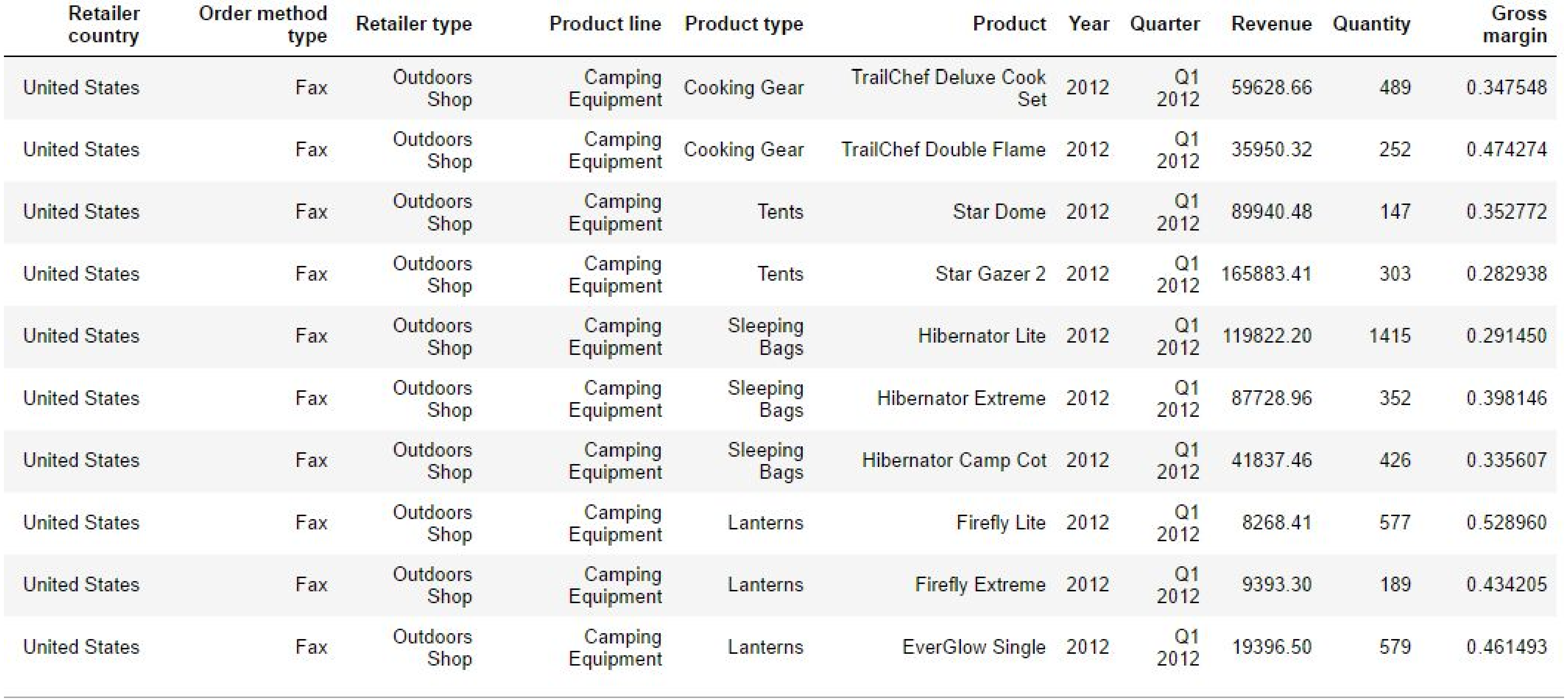


Źródło: http://prog3.com/article/2015-06-18/2824958

Działa wszędzie (*)

- Bazy relacyjne
  - tradycyjne, np.: MySQL, PostgreSQL, Oracle, SQL Server
  - skalowalne, np.: CockroachDB, AWS Aurora, Azure SQL Server, Google Cloud Spanner
- Hurtownie danych
  - tradycyjne, np.: AWS Redshift, Vertica, Teradata, Azure SQL Data Ware
  - zbiornik danych, np: Hive, Druid, Presto, Google BigQuery, AWS Athena, Azure Data Lake Analytics
- Silniki obliczeniowe
  - rozproszone, np.: Spark, Flink
- Bazy nierelacyjne (NoSQL) (*)
  - kolumnowe, np.: Phoenix (Hbase), Cassandra (CQL)
  - klucz-wartość: Aerospike (AQL)
  - przez klaster, np: Spark

Przykładowe dane

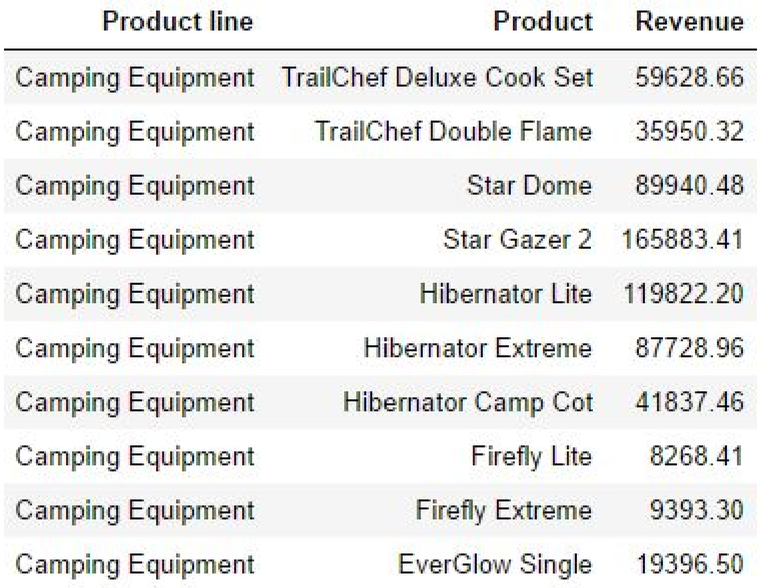


Retailer country	Order method type	Retailer type	Product line	Product type	Product	Year	Quarter	Revenue	Quantity	Gross margin
United States	Fax	Outdoors Shop	Camping Equipment	Cooking Gear	TrailChef Deluxe Cook Set	2012	Q1 2012	59628.66	489	0.347548
United States	Fax	Outdoors Shop	Camping Equipment	Cooking Gear	TrailChef Double Flame	2012	Q1 2012	35950.32	252	0.474274
United States	Fax	Outdoors Shop	Camping Equipment	Tents	Star Dome	2012	Q1 2012	89940.48	147	0.352772
United States	Fax	Outdoors Shop	Camping Equipment	Tents	Star Gazer 2	2012	Q1 2012	165883.41	303	0.282938
United States	Fax	Outdoors Shop	Camping Equipment	Sleeping Bags	Hibernator Lite	2012	Q1 2012	119822.20	1415	0.291450
United States	Fax	Outdoors Shop	Camping Equipment	Sleeping Bags	Hibernator Extreme	2012	Q1 2012	87728.96	352	0.398146
United States	Fax	Outdoors Shop	Camping Equipment	Sleeping Bags	Hibernator Camp Cot	2012	Q1 2012	41837.46	426	0.335607
United States	Fax	Outdoors Shop	Camping Equipment	Lanterns	Firefly Lite	2012	Q1 2012	8268.41	577	0.528960
United States	Fax	Outdoors Shop	Camping Equipment	Lanterns	Firefly Extreme	2012	Q1 2012	9393.30	189	0.434205
United States	Fax	Outdoors Shop	Camping Equipment	Lanterns	EverGlow Single	2012	Q1 2012	19396.50	579	0.461493

Źródło: https://www.ibm.com/communities/analytics/watson-analytics-blog/sales-products-sample-data/

Prosty przykład… (SQL)

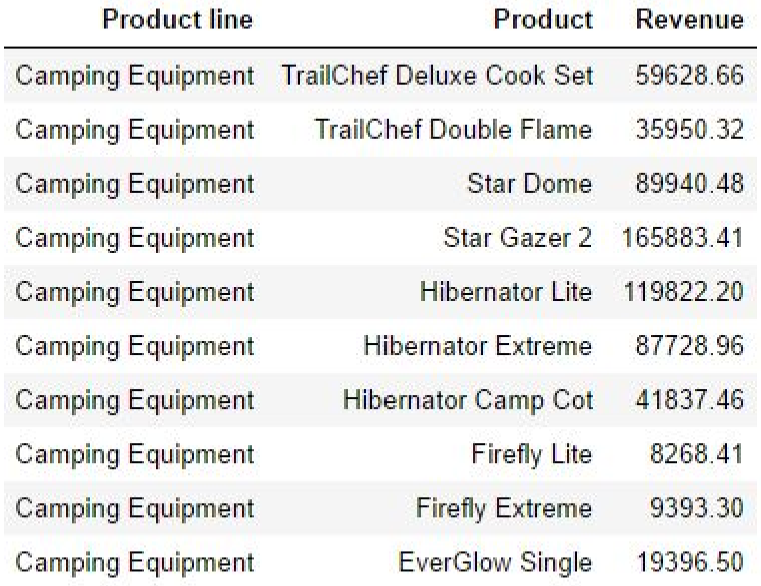
```sql
SELECT
    `Product line`,
    Product,
    Revenue
FROM sales
WHERE Year = 2012
LIMIT 10
```



Product line	Product	Revenue
Camping Equipment	TrailChef Deluxe Cook Set	59628.66
Camping Equipment	TrailChef Double Flame	35950.32
Camping Equipment	Star Dome	89940.48
Camping Equipment	Star Gazer 2	165883.41
Camping Equipment	Hibernator Lite	119822.20
Camping Equipment	Hibernator Extreme	87728.96
Camping Equipment	Hibernator Camp Cot	41837.46
Camping Equipment	Firefly Lite	8268.41
Camping Equipment	Firefly Extreme	9393.30
Camping Equipment	EverGlow Single	19396.50

Prosty przykład… (Scala)

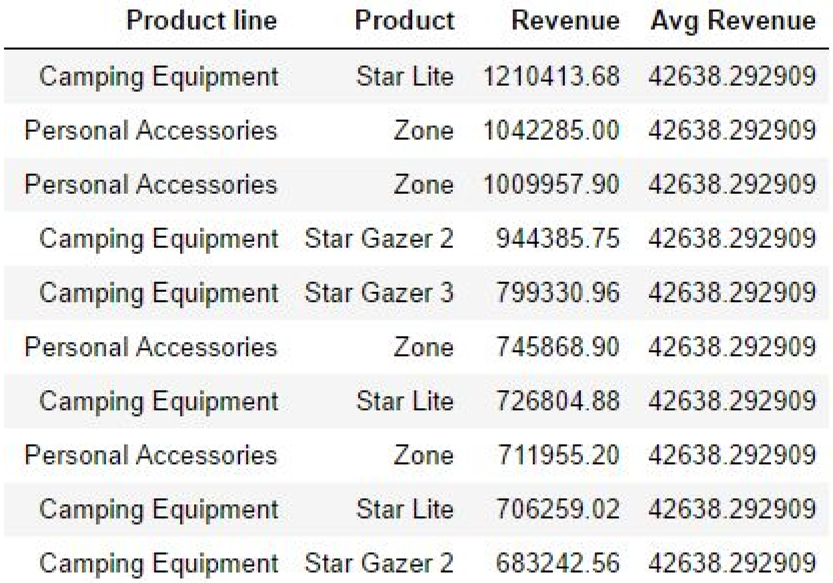
```
var i = 0
val results = new ListBuffer[Result1]()
breakable {
    for (r <- salesRows) {
        if (r.year == 2012) {
            i += 1
            results.append(
                Result1(r.productLine,
                        r.product,
                        r.revenue))
        }

        if (i >= 10)
            break
    }
}
results.toList
```



Product line	Product	Revenue
Camping Equipment	TrailChef Deluxe Cook Set	59628.66
Camping Equipment	TrailChef Double Flame	35950.32
Camping Equipment	Star Dome	89940.48
Camping Equipment	Star Gazer 2	165883.41
Camping Equipment	Hibernator Lite	119822.20
Camping Equipment	Hibernator Extreme	87728.96
Camping Equipment	Hibernator Camp Cot	41837.46
Camping Equipment	Firefly Lite	8268.41
Camping Equipment	Firefly Extreme	9393.30
Camping Equipment	EverGlow Single	19396.50

… ale czy na pewno (SQL)

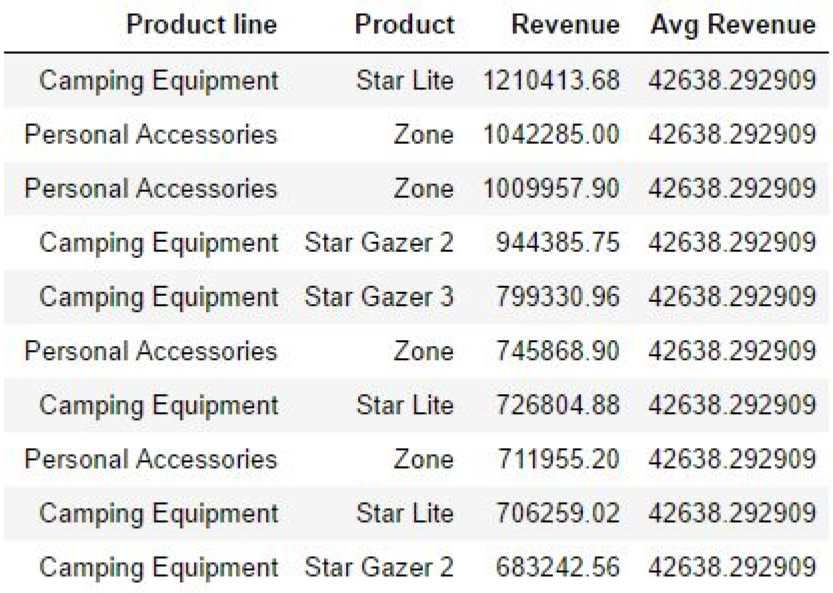
```
SELECT
    `Product line`,
    Product,
    Revenue,
    (SELECT AVG(Revenue) FROM sales)
         AS `Avg Revenue`
FROM sales
WHERE Year = 2012
ORDER BY Revenue DESC
LIMIT 10
```



Product line	Product	Revenue	Avg Revenue
Camping Equipment	Star Lite	1210413.68	42638.292909
Personal Accessories	Zone	1042285.00	42638.292909
Personal Accessories	Zone	1009957.90	42638.292909
Camping Equipment	Star Gazer 2	944385.75	42638.292909
Camping Equipment	Star Gazer 3	799330.96	42638.292909
Personal Accessories	Zone	745868.90	42638.292909
Camping Equipment	Star Lite	726804.88	42638.292909
Personal Accessories	Zone	711955.20	42638.292909
Camping Equipment	Star Lite	706259.02	42638.292909
Camping Equipment	Star Gazer 2	683242.56	42638.292909

... ale czy na pewno (Scala)

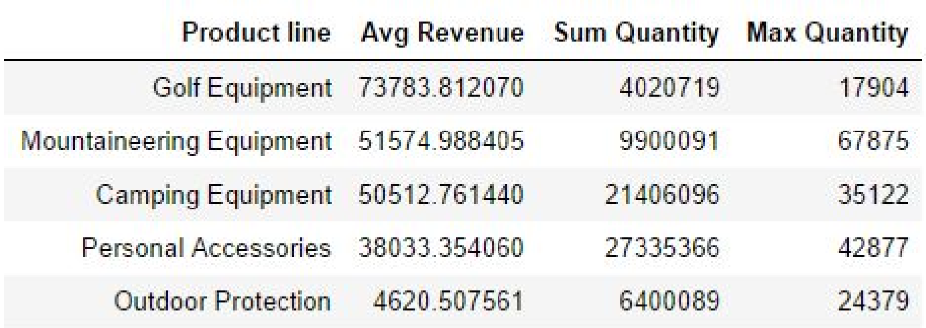
```scala
val sorted = salesRows.sortBy(- _.revenue)
val results1 = select1(sorted)
var revenueSum = 0.0
for (r <- salesRows) {
    revenueSum += r.revenue
}
val avgRevenue = revenueSum/salesRows.size
val results2 = new ListBuffer[Result2]()
for (r <- results1) {
    results2.append(
        Result2(r.productLine,
                r.product,
                r.revenue,
                avgRevenue
            )
    )
}
results2.toList
```



Product line	Product	Revenue	Avg Revenue
Camping Equipment	Star Lite	1210413.68	42638.292909
Personal Accessories	Zone	1042285.00	42638.292909
Personal Accessories	Zone	1009957.90	42638.292909
Camping Equipment	Star Gazer 2	944385.75	42638.292909
Camping Equipment	Star Gazer 3	799330.96	42638.292909
Personal Accessories	Zone	745868.90	42638.292909
Camping Equipment	Star Lite	726804.88	42638.292909
Personal Accessories	Zone	711955.20	42638.292909
Camping Equipment	Star Lite	706259.02	42638.292909
Camping Equipment	Star Gazer 2	683242.56	42638.292909

Grupowanie

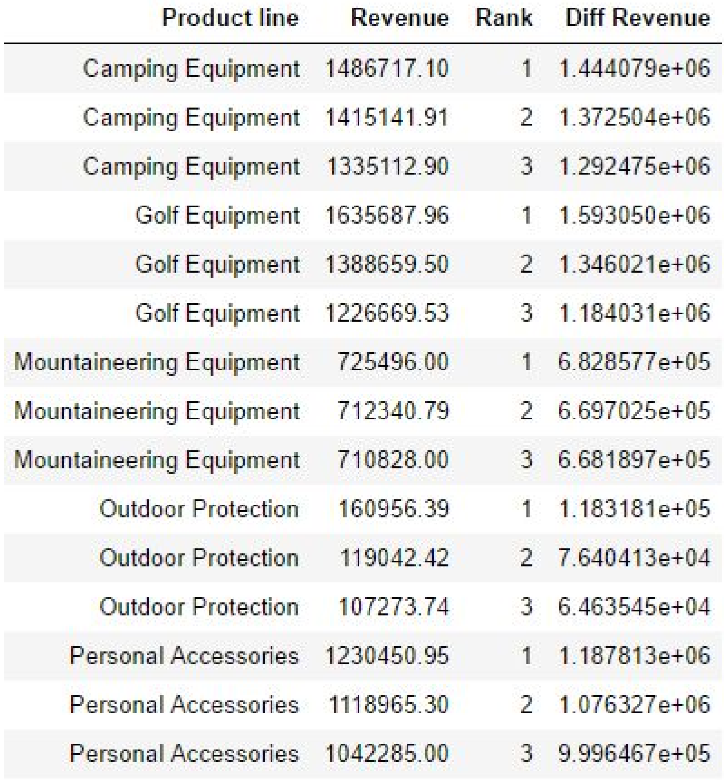
```
SELECT
    `Product line`,
    AVG(Revenue) AS `Avg Revenue`,
    SUM(Quantity) AS `Sum Quantity`,
    MAX(Quantity) AS `Max Quantity`
FROM sales
GROUP BY `Product line`
ORDER BY `Avg Revenue` DESC
```



Product line	Avg Revenue	Sum Quantity	Max Quantity
Golf Equipment	73783.812070	4020719	17904
Mountaineering Equipment	51574.988405	9900091	67875
Camping Equipment	50512.761440	21406096	35122
Personal Accessories	38033.354060	27335366	42877
Outdoor Protection	4620.507561	6400089	24379

Funkcje okienne (Window functions)

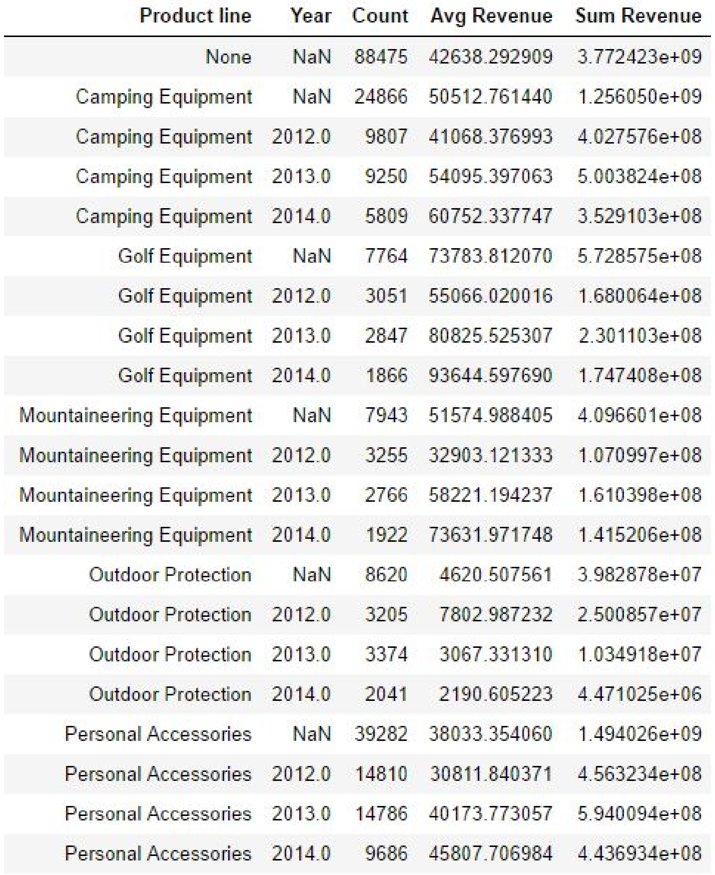
```
SELECT  *
FROM (
 SELECT
   `Product line`,
   `Revenue`,
   RANK() OVER(
     PARTITION BY `Product line` ORDER BY Revenue DESC
     ) AS Rank,
   (Revenue - (SELECT AVG(Revenue) FROM sales))
     AS `Diff Revenue`
  FROM sales
  )
WHERE Rank <= 3
ORDER BY `Product line`
```



Product line	Revenue	Rank	Diff Revenue
Camping Equipment	1486717.10	1	1.444079e+06
Camping Equipment	1415141.91	2	1.372504e+06
Camping Equipment	1335112.90	3	1.292475e+06
Golf Equipment	1635687.96	1	1.593050e+06
Golf Equipment	1388659.50	2	1.346021e+06
Golf Equipment	1226669.53	3	1.184031e+06
Mountaineering Equipment	725496.00	1	6.828577e+05
Mountaineering Equipment	712340.79	2	6.697025e+05
Mountaineering Equipment	710828.00	3	6.681897e+05
Outdoor Protection	160956.39	1	1.183181e+05
Outdoor Protection	119042.42	2	7.640413e+04
Outdoor Protection	107273.74	3	6.463545e+04
Personal Accessories	1230450.95	1	1.187813e+06
Personal Accessories	1118965.30	2	1.076327e+06
Personal Accessories	1042285.00	3	9.996467e+05

ROLLUP

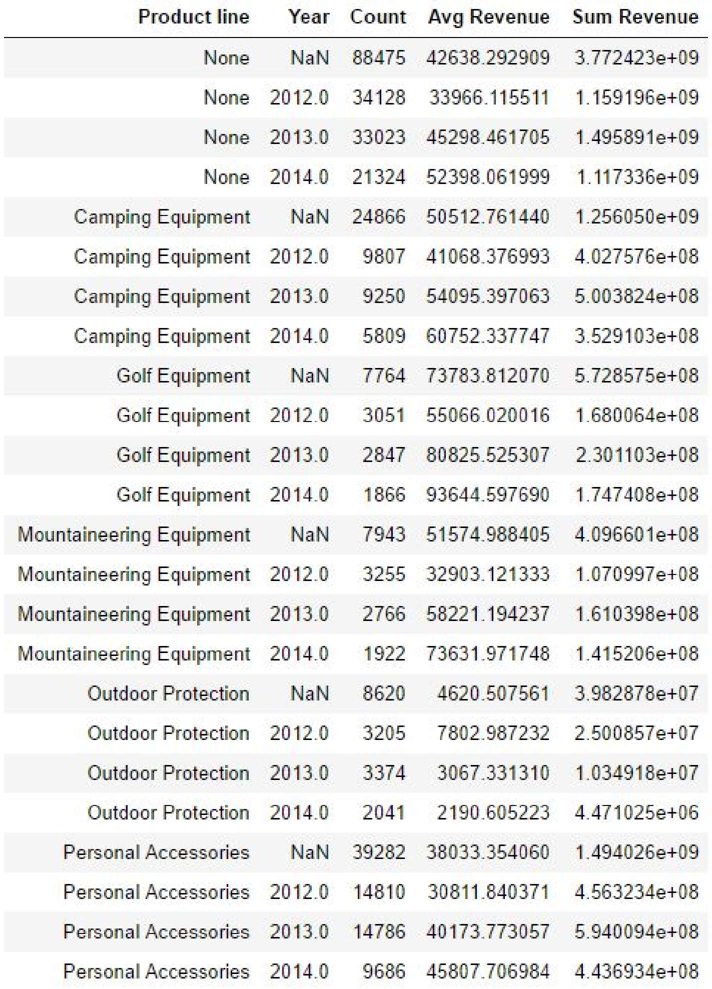
```
SELECT
    `Product line`,
    Year,
    COUNT(*) AS Count,
    AVG(Revenue) AS `Avg Revenue`,
    SUM(Revenue) AS `Sum Revenue`
FROM sales
GROUP BY `Product line`, Year WITH ROLLUP
ORDER BY `Product line`, Year
```



Product line	Year	Count	Avg Revenue	Sum Revenue
None	NaN	88475	42638.292909	3.772423e+09
Camping Equipment	NaN	24866	50512.761440	1.256050e+09
Camping Equipment	2012.0	9807	41068.376993	4.027576e+08
Camping Equipment	2013.0	9250	54095.397063	5.003824e+08
Camping Equipment	2014.0	5809	60752.337747	3.529103e+08
Golf Equipment	NaN	7764	73783.812070	5.728575e+08
Golf Equipment	2012.0	3051	55066.020016	1.680064e+08
Golf Equipment	2013.0	2847	80825.525307	2.301103e+08
Golf Equipment	2014.0	1866	93644.597690	1.747408e+08
Mountaineering Equipment	NaN	7943	51574.988405	4.096601e+08
Mountaineering Equipment	2012.0	3255	32903.121333	1.070997e+08
Mountaineering Equipment	2013.0	2766	58221.194237	1.610398e+08
Mountaineering Equipment	2014.0	1922	73631.971748	1.415206e+08
Outdoor Protection	NaN	8620	4620.507561	3.982878e+07
Outdoor Protection	2012.0	3205	7802.987232	2.500857e+07
Outdoor Protection	2013.0	3374	3067.331310	1.034918e+07
Outdoor Protection	2014.0	2041	2190.605223	4.471025e+06
Personal Accessories	NaN	39282	38033.354060	1.494026e+09
Personal Accessories	2012.0	14810	30811.840371	4.563234e+08
Personal Accessories	2013.0	14786	40173.773057	5.940094e+08
Personal Accessories	2014.0	9686	45807.706984	4.436934e+08

CUBE

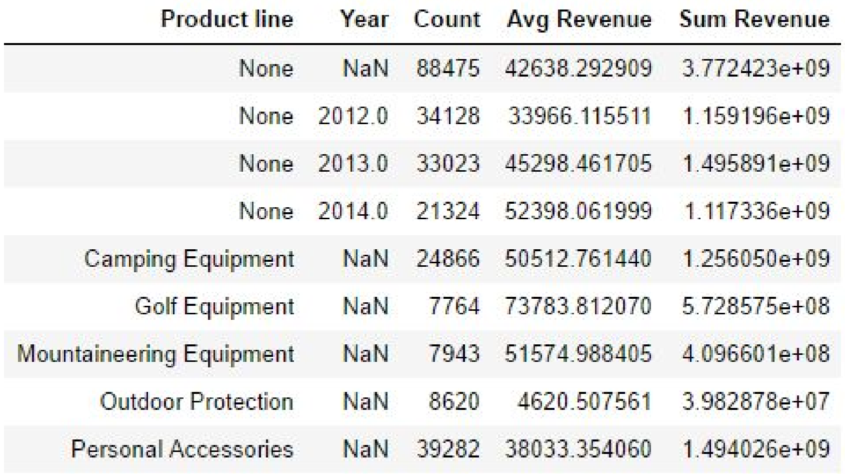
```
SELECT
    `Product line`,
    Year,
    COUNT(*) AS Count,
    AVG(Revenue) AS `Avg Revenue`,
    SUM(Revenue) AS `Sum Revenue`
FROM sales
GROUP BY `Product line`, Year WITH CUBE
ORDER BY `Product line`, Year
```



Product line	Year	Count	Avg Revenue	Sum Revenue
None	NaN	88475	42638.292909	3.772423e+09
None	2012.0	34128	33966.115511	1.159196e+09
None	2013.0	33023	45298.461705	1.495891e+09
None	2014.0	21324	52398.061999	1.117336e+09
Camping Equipment	NaN	24866	50512.761440	1.256050e+09
Camping Equipment	2012.0	9807	41068.376993	4.027576e+08
Camping Equipment	2013.0	9250	54095.397063	5.003824e+08
Camping Equipment	2014.0	5809	60752.337747	3.529103e+08
Golf Equipment	NaN	7764	73783.812070	5.728575e+08
Golf Equipment	2012.0	3051	55066.020016	1.680064e+08
Golf Equipment	2013.0	2847	80825.525307	2.301103e+08
Golf Equipment	2014.0	1866	93644.597690	1.747408e+08
Mountaineering Equipment	NaN	7943	51574.988405	4.096601e+08
Mountaineering Equipment	2012.0	3255	32903.121333	1.070997e+08
Mountaineering Equipment	2013.0	2766	58221.194237	1.610398e+08
Mountaineering Equipment	2014.0	1922	73631.971748	1.415206e+08
Outdoor Protection	NaN	8620	4620.507561	3.982878e+07
Outdoor Protection	2012.0	3205	7802.987232	2.500857e+07
Outdoor Protection	2013.0	3374	3067.331310	1.034918e+07
Outdoor Protection	2014.0	2041	2190.605223	4.471025e+06
Personal Accessories	NaN	39282	38033.354060	1.494026e+09
Personal Accessories	2012.0	14810	30811.840371	4.563234e+08
Personal Accessories	2013.0	14786	40173.773057	5.940094e+08
Personal Accessories	2014.0	9686	45807.706984	4.436934e+08

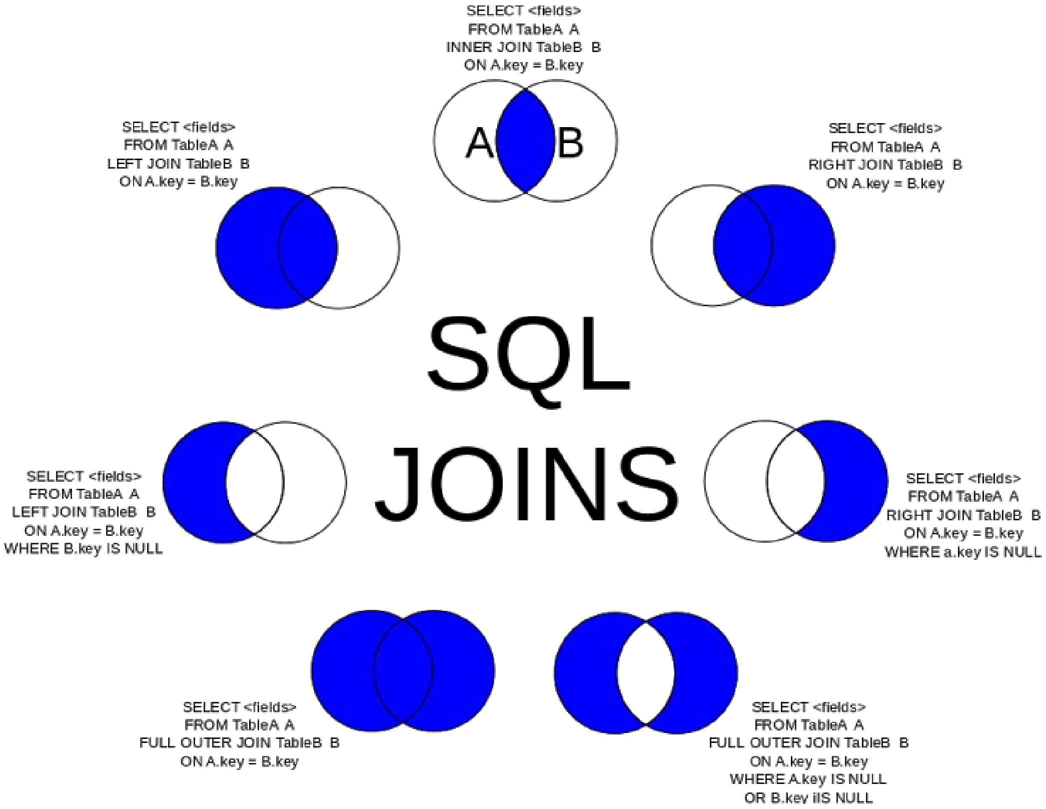
GROUPING SETS

```
SELECT
    `Product line`,
    Year,
    COUNT(*) AS Count,
    AVG(Revenue) AS `Avg Revenue`,
    SUM(Revenue) AS `Sum Revenue`
FROM sales
GROUP BY `Product line`, Year
GROUPING SETS (`Product line`, Year, ())
ORDER BY `Product line`, Year
```



Product line	Year	Count	Avg Revenue	Sum Revenue
None	NaN	88475	42638.292909	3.772423e+09
None	2012.0	34128	33966.115511	1.159196e+09
None	2013.0	33023	45298.461705	1.495891e+09
None	2014.0	21324	52398.061999	1.117336e+09
Camping Equipment	NaN	24866	50512.761440	1.256050e+09
Golf Equipment	NaN	7764	73783.812070	5.728575e+08
Mountaineering Equipment	NaN	7943	51574.988405	4.096601e+08
Outdoor Protection	NaN	8620	4620.507561	3.982878e+07
Personal Accessories	NaN	39282	38033.354060	1.494026e+09

Złączenie (JOIN)



```
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL
```

```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE a.key IS NULL
```

```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.kev iIS NULL
```

Źródło: http://bailiwick.io/2015/07/13/joining-data-frames-in-spark-sql/

Partycjonowanie

```
-- Hive (0.13+)
CREATE TABLE my_table (key BIGINT, value TEXT)
PARTITIONED BY (`date` DATE)
COMMENT 'My Table with Partitions'
STORED AS PARQUET
```

```
$ hdfs dfs -ls -h /user/hive/warehouse/my_table

-rw-r--r--  3 jakub hive  0 2017-01-01 01:01 /user/hive/warehouse/my_table/_SUCCESS

drwxr-xr-x  - jakub hive  0 2017-01-01 01:01 /user/hive/warehouse/my_table/date=2016-12-01

drwxr-xr-x  - jakub hive  0 2017-01-01 01:01 /user/hive/warehouse/my_table/date=2016-12-02

drwxr-xr-x  - jakub hive  0 2017-01-01 01:01 /user/hive/warehouse/my_table/date=2016-12-03
```

Wady SQL też są

- Nie ma zmiennych
  - Są aliasy ale nie wszędzie jednakowo wspierane
  - Piekło zagnieżdżeń
- Zapytanie jest tekstem
  - Nie ma za bardzo analizy statycznej
  - Większość problemów wychodzi dopiero po uruchomieniu
- Różne podejście do standardów
- Dość płaska struktura tabeli

```
SELECT  *
FROM (
 SELECT
   `Product line`,
   `Revenue`,
   RANK() OVER(
     PARTITION BY `Product line` ORDER BY Revenue DESC
     ) AS Rank,
   (Revenue - (SELECT AVG(Revenue) FROM sales))
     AS `Diff Revenue`
  FROM sales
  )
WHERE Rank <= 3
ORDER BY `Product line`
```

Programowanie funkcyjne (Scala)

- Nie jest deklaratywne, zatem wszystko robimy sami
- Operujemy wyłącznie na wierszach
- Nie ma pojęcia kolumn
- Do wielu operacja musimy przekształcać dane i wykonywać operacje na parach klucz-wartość
- To jak się robi złączenie (JOIN) używając tylko *map* i *reduce*? Niezbyt prosto...

```
val sumRevenue = salesRows
    .map(_.revenue)
    .sum
val avgRevenue = sumRevenue / salesRows.size
salesRows.filter(_.year == 2012)
    .sortBy(- _.revenue)
    .map(r =>
        Result2(r.productLine,
                r.product,
                r.revenue,
                avgRevenue)
    ).take(10)
```

Data Frame

- Obiekt opakowujący tabele
  - Podział na wiersze i kolumny
- Współczesna koncepcja zaproponowana w języku R
  - Sam język R powstał w 1993 r.
- Obróbka danych inspirowana SQL, ale z różnym poziomem zgodności
  - Część projektów pozwala na zapytania SQL, ale nie wszystkie
- Jako, że Data Frame to obiekt, pozwala na lepszą analizę statyczną przed uruchomieniem
- Data Frame jest różnie implementowany, więc nie zawsze działa dokładnie tak samo, ale jest bardzo zbliżony do SQL

Data Frame

```
SELECT  *
FROM (
 SELECT
   `Product line`,
   `Revenue`,
   RANK() OVER(
     PARTITION BY `Product line` ORDER BY Revenue DESC
     ) AS Rank,
   (Revenue - (SELECT AVG(Revenue) FROM sales))
     AS `Diff Revenue`
  FROM sales
  )
WHERE Rank <= 3
ORDER BY `Product line`
```

```
// Spark 2+
val w = Window.partitionBy("Product line")
    .orderBy($"Revenue".desc)

val df = sales.select(
        $"Product line",
        $"Revenue",
        rank.over(w).as("Rank"),
        ($"Revenue" - expr("(SELECT AVG(Revenue)
FROM sales)")).as("Diff Revenue")
    ).where("Rank <= 3")
df.show()
```

Projekty z pochodnymi Data Frame

Języki i projekty implementujące Data Frame:

- R
  - Wbudowane w język (https://www.r-project.org/)
  - dplyr (http://dplyr.tidyverse.org/)
- Python
  - Pandas (http://pandas.pydata.org/)
  - Dask (http://dask.pydata.org/)
  - Spark (http://spark.apache.org/)
- Java/Scala
  - Spark (http://spark.apache.org/)
  - Flink (https://flink.apache.org/)
  - jOOQ (https://www.jooq.org/)

Dziękuję!
Pytania?

Kod zapytań dostępny na: https://github.com/jsnowacki/why-sql-talk-demo