Nowa składnia zapytań SQL w BigQuery

Pipe Query Syntax

Andrzej Kukuła
andrzej.kukula@datacommunity.pl

Miasto Bielsko-Biała

SQL Day Lite

GOLD SPONSORS

Woodler

DATUMO

SILVER SPONSOR

Xebia

PARTNERS

Uniwersytet Bielsko-Bialski

RESET KOŁO INFORMATYCZNE UBB WBMII

Agenda

- What is BigQuery
- SQL
- Pipe Query Syntax
- Demos

What is BigQuery?

BigQuery

- Modern, AI-era data platform
- Managed data warehouse / data lakehouse
  - Cloud native
  - Enterprise-grade
  - Distributed
  - Exabyte scale
- Fully serverless from ground up
  - No infrastructure to manage
- Batch and streaming processing
  - Continuous queries
- Supports AI natively
  - Generative AI, pretrained models
  - Agents
  - Machine learning
  - Time series
  - Augmented analytics

BigQuery

- Efficient columnar storage (Capacitor) for native tables
- Supports popular formats
  - Iceberg, Delta Lake
  - Parquet, ORC, AVRO, (ND)JSON, CSV
- SQL at the core - GoogleSQL - very rich in features
- 2 pricing models
  - On-demand
  - Capacity-based
- Cross-cloud (BigQuery Omni)
- SDKs and clients for many programming languages
- Rich integrations
- Scheduling
- Notebooks
- ... and more

What is BigQuery good for?

Do use it for:

- Storing lots of structured data natively
- Data warehouses, data lakes
- Fully-structured data (tables, views, datasets)
- Semi-structured data (JSON),
- Unstructured data
  - Through lakes or object tables from storage
- Denormalized data model
- Analytical queries (on large amounts of data)
- Queries for reporting
- Streaming and batch loads
- CDC, ELT, ETL
- AI tasks (data understanding, exploration, agents, image analysis, document analysis, embeddings generation, vector search)
  - Consider other products for specialized AI needs

Don't use it for:

- OLTP workload
- High-frequency or high volume single-row queries
- Ultra low latency data access (sub-millisecond)
- Primary storage for unstructured data
  - Not in native tables
- Lots of joins in queries
- Normalized data model
- Real-time compute-heavy stream processing engine
- Use Cloud Dataflow for that
- Drop-in replacement for PostgreSQL, MySQL, SQL Server, SQLite etc.
  - BigQuery has SQL but it's not a RDBMS
- Scaffolding for a spreadsheet-based BI solutions

SQL
Structured Query Language

SQL

- De facto standard *) query language
  - Works in hundreds of engines / products
- Declarative: emphasizes what needs to be returned
  - Not how to fetch and process rows of data
  - Hides complexity of data storage, retrieval, manipulation and modification
- Set-based operations
- Astonishing amount of features
- Innovating at high speed
- Best for > 95% *) data processing tasks
- Handles big data with ease
- Works great with AI
- Not going anywhere

SQL syntax

```
SELECT a, b, SUM(c) AS s,
  RANK() OVER (ORDER BY d) AS r
FROM t
INNER JOIN u USING (id)
WHERE b < 100
GROUP BY a, b
HAVING SUM(c) > 10
QUALIFY r <= 3
ORDER BY a, b
LIMIT 10
```

SQL

- Arbitrary and rigid syntax
- Quite difficult to learn
- Queries are not linearly readable
  - Inside-out flow, starting with tables or subqueries in the middle
- Distant side effects
  - e.g. aggregates and GROUP BY
  - Window functions
- Redundant clauses (e.g. WHERE, HAVING, QUALIFY, PREWHERE)
- Many operations need subqueries
- Verbose and repetitive
  - List the same columns in SELECT, GROUP BY, ORDER BY, and in subqueries
- Difficult to implement autocompletion

SQL syntax	Semantic evaluation order

`SELECT a, b, SUM(c) AS s,`
`  RANK() OVER (ORDER BY d) AS r`
`FROM t`
`INNER JOIN u USING (id)`
`WHERE b < 100`
`GROUP BY a, b`
`HAVING SUM(c) > 10`
`QUALIFY r <= 3`
`ORDER BY a, b`
`LIMIT 10`
	from
	join
	filter
	aggregate
	filter
	sort
	window
	filter
	limit
	project

Pipeline processing

Pipeline features

- Sequential transformations
- Improved readability
- Modularity and reusability

Shell languages - the originators

Specialized query languages

- KQL, SPL

Functional languages

- OCaml, R, F#, Elixir, Haskell

Method chaining (fluent syntax)

- JavaScript, Python, Ruby, Java, C#, Scala, Rust
- Python data frames
- C++20 (std::views)

(bash)

```bash
curl -s https://static.nbp.pl/.../archiwum_tab_a_2025.csv  \
  | grep ^2025     \  # filter data
  | cut -d";" -f3  \  # transform (extract 'USD' field)
  | sed 's/,/./'   \  # transform (change decimal point)
  | sort -n -r     \  # sort data
  | head -n 1         # get the required item
```

(JavaScript)

```javascript
const highestUsdRate = csvFile
  .split(/\r?\n/)
  .filter(line => line.startsWith('2025'))
  .map(line => line.split(';')[2])
  .map(rateString => rateString.replace(',', '.'))
  .map(rateString => parseFloat(rateString))
  .sort((a, b) => b - a)
  .shift();  // or [0]

  // (I know it's not optimal 🙂)
```

But... we cannot fix SQL...
... can we?

SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL

Jeff Shute
Google, Inc.

Shannon Bales
Google, Inc.

Matthew Brown
Google, Inc.

Jean-Daniel Browne
Google, Inc.

Brandon Dolphin
Google, Inc.

Romit Kudtarkar
Google, Inc.

Andrey Litvinov
Google, Inc.

Jingchi Ma
Google, Inc.

John Morcos
Google, Inc.

Michael Shen
Google, Inc.

David Wilhite
Google, Inc.

Xi Wu
Google, Inc.

Lulan Yu
Google, Inc.

sql-pipes-paper@google.com

https://research.google/pubs/sql-has-problems-we-can-fix-them-pipe-syntax-in-sql/

Pipe Query Syntax in BigQuery

Pipe |> Query |> Syntax

What is it

- Extension to GoogleSQL
- Does not replace SQL - extends it
- Very straightforward, easy to learn
- Blends with traditional syntax
- Natural operator order
- Sequential execution
- Improves readability
- Iterative query development
- Enables easier optimization
- Doesn't deprecate any GoogleSQL feature

SQL syntax	Pipe query syntax

`SELECT a, b, SUM(c) AS s,`
`  RANK() OVER (ORDER BY d) AS r`
`FROM t`
`JOIN u USING (id)`
`WHERE b < 100`
`GROUP BY a, b`
`HAVING SUM(c) > 0`
`ORDER BY a, b`
`LIMIT 10`
	`FROM t`
	`|> JOIN u USING (id)`
	`|> WHERE b < 100`
	`|> AGGREGATE SUM(c) AS s`
	`  GROUP BY a, b`
	`|> WHERE s > 0`
	`|> ORDER BY a, b`
	`|> LIMIT 10`
	`|> SELECT a, b, s,`
	`  RANK() OVER (ORDER BY d) AS r`

Pipe |> Query |> Syntax

Key characteristics

- Simple syntax: pipe symbol, operator name, arguments:

```
|> operator_name argument_list
```

- Pipe operators can be added to the end of any valid query
- Pipe operators can be applied in any order, any number of times
- Pipe syntax works anywhere standard syntax is supported: queries, views, table-valued functions etc.

- The simplest query:

```
FROM data_source;
```

```
|> SELECT
|> EXTEND
|> SET
|> DROP
|> RENAME
|> AS

|> WHERE
|> ORDER BY
|> LIMIT

|> AGGREGATE

|> JOIN
|> UNION
|> INTERSECT
|> EXCEPT

|> WITH

|> PIVOT
|> UNPIVOT
|> CALL
|> TABLESAMPLE
```

Pipe |> Query |> Syntax

Pipe operator semantics

- Each pipe operator performs a self-contained operation
- A pipe operator consumes the input rowset passed to it through the pipe symbol `|>` and produces a new rowset as output
  - Semantically, not physically
- A pipe operator can reference only columns available in its immediate input rowset(s)
  - Columns from earlier operators in the same query aren't visible (semantically)

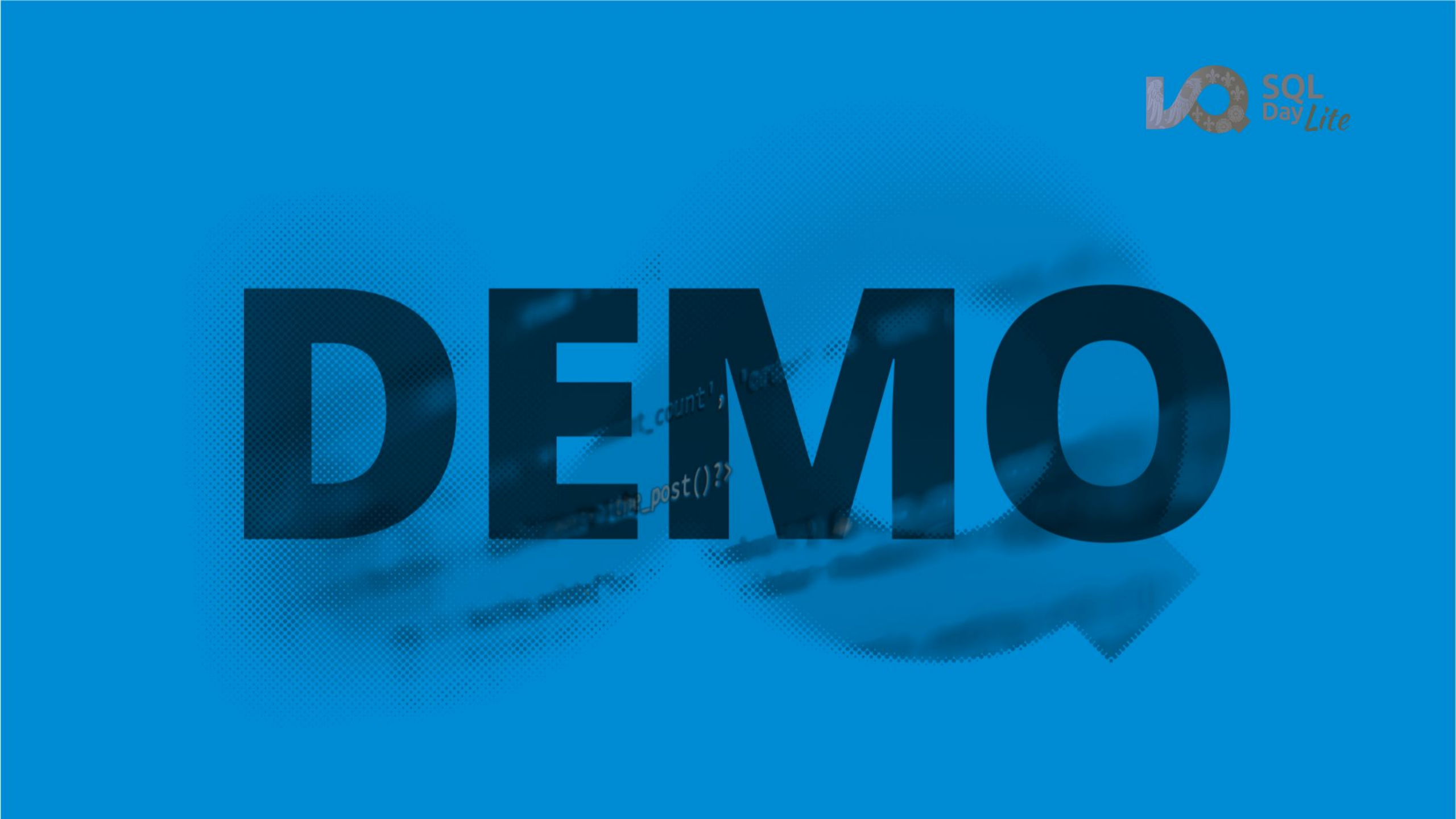
```
|> SELECT
|> EXTEND
|> SET
|> DROP
|> RENAME
|> AS

|> WHERE
|> ORDER BY
|> LIMIT

|> AGGREGATE

|> JOIN
|> UNION
|> INTERSECT
|> EXCEPT

|> WITH

|> PIVOT
|> UNPIVOT
|> CALL
|> TABLESAMPLE
```

Who's it for? Everyone!

- **SQL beginners**
  - Very easy to learn
- **SQL experts**
  - Same operators and similar syntax, better structure
  - Much more productive
- **SQL dislikers**
  - Fixes many difficult or annoying parts that confuse users or cause resistance

Who's it for? Everyone!

- **No migrations, nothing to lose**
  - Existing queries all still work
  - New syntax can be introduced incrementally
  - No translations needed
- **Fully supported by Google**
  - GA in BigQuery as of April 2025
- **Market is starting to adopt it**
  - Apache Spark SQL
  - Databricks SQL
  - DuckDB (partial, through extension)
  - Some products have open FRs about it


DEMO
SQL Day Lite

Data Community

```
FROM `me`
|> THANK YOU
```