OST
Eastern Switzerland
University of Applied Sciences

Automated Database Benchmarking Tool

Performance Analysis of MySQL, PostgreSQL and Neo4j using Different Data Scenarios

Institute	Eastern Switzerland University of Applied Science
Program	MSE Computer Science
Module	DB Seminar
Author	Roman Bögli
Supervisor	Prof. Stefan F. Keller
Date	14. April 2022
Context	Final Presentation

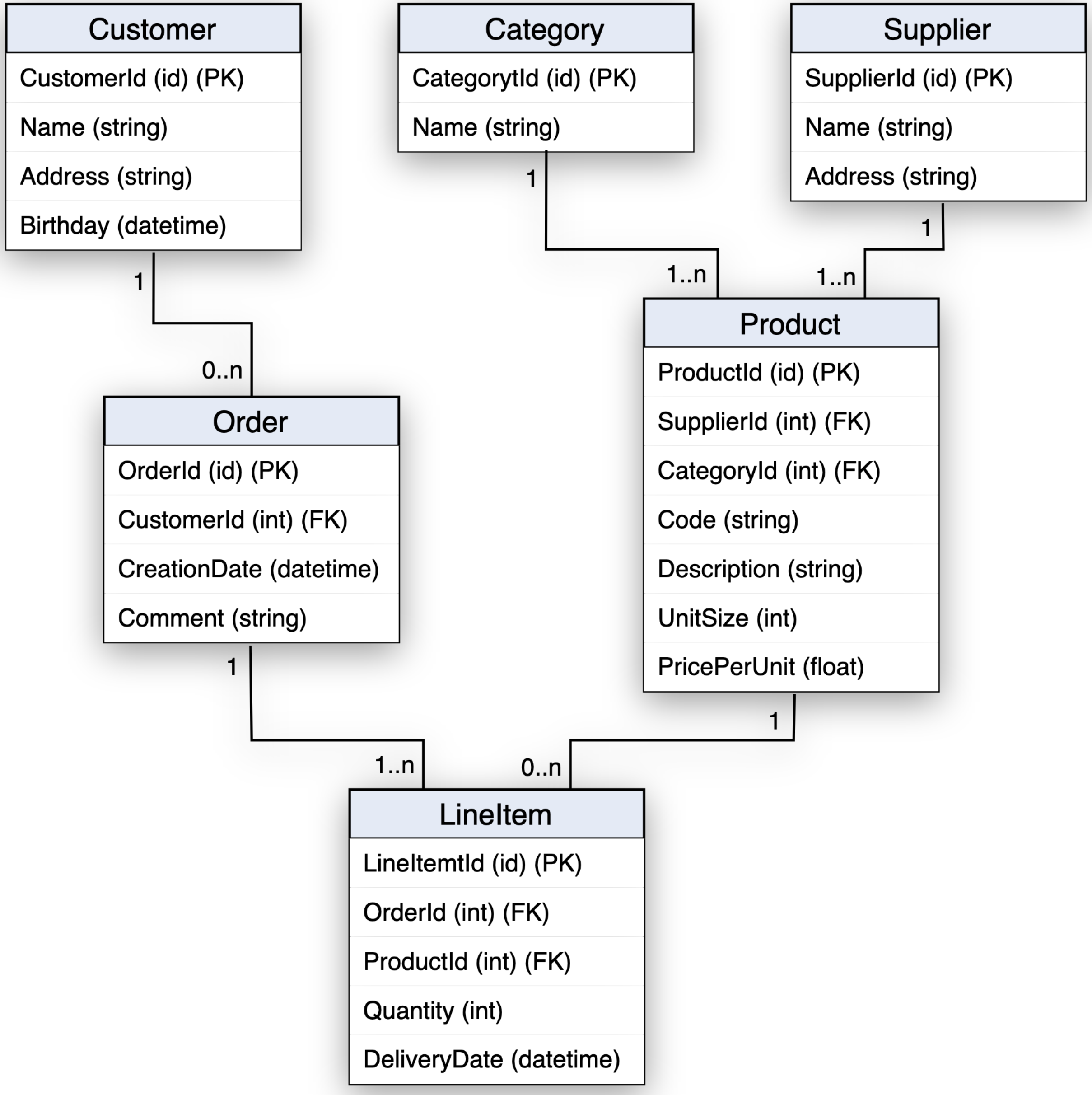
also available as PDF

Content

- Relational DBMS vs. Graph-Based DBMS
- Tool `godbbench`
- Synthetic Script & Substitution
- Custom Scripts (`merchant`, `employees`)
- Automation
- Result Analysis
- Conclusion & Future Work

Relational DBMS

- Tables are entities
- Relationships using keys
- Homogenous data through schema
- Ideal for **predefinable** & **rigid** data use cases



Customer
- CustomerId (id) (PK)
- Name (string)
- Address (string)
- Birthday (datetime)

Category
- CategoryId (id) (PK)
- Name (string)

Supplier
- SupplierId (id) (PK)
- Name (string)
- Address (string)

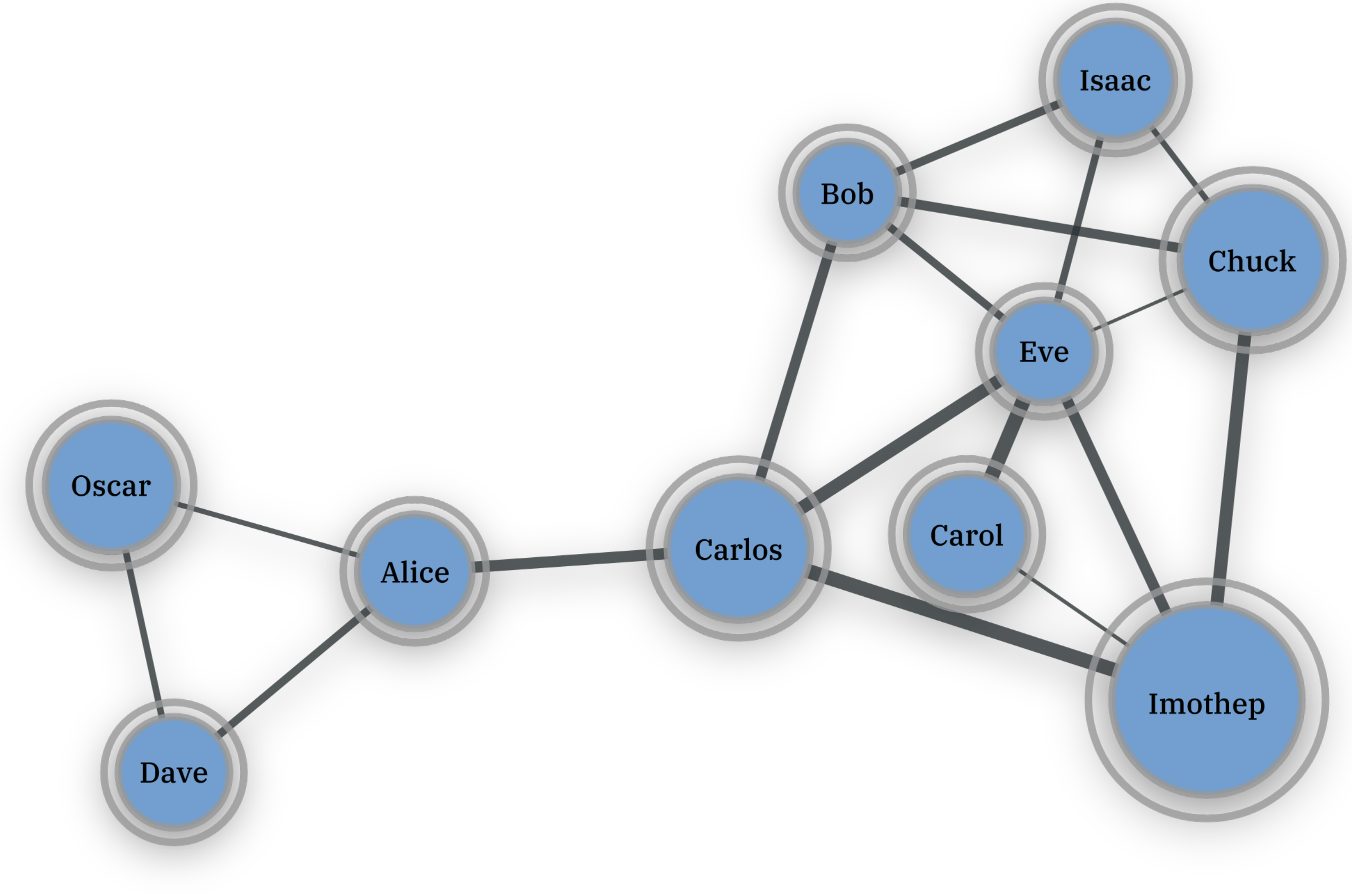
Order
- OrderId (id) (PK)
- CustomerId (int) (FK)
- CreationDate (datetime)
- Comment (string)

Product
- ProductId (id) (PK)
- SupplierId (int) (FK)
- CategoryId (int) (FK)
- Code (string)
- Description (string)
- UnitSize (int)
- PricePerUnit (float)

LineItem
- LineItemtId (id) (PK)
- OrderId (int) (FK)
- ProductId (int) (FK)
- Quantity (int)
- DeliveryDate (datetime)

Graph-Based DBMS

- Attributed nodes and edges
- Relationships are first-class citizen
- Heterogenous data (schema-less)
- Ideal for **alternating** & **highly connected** data use cases



Query Languages

Query adult customers

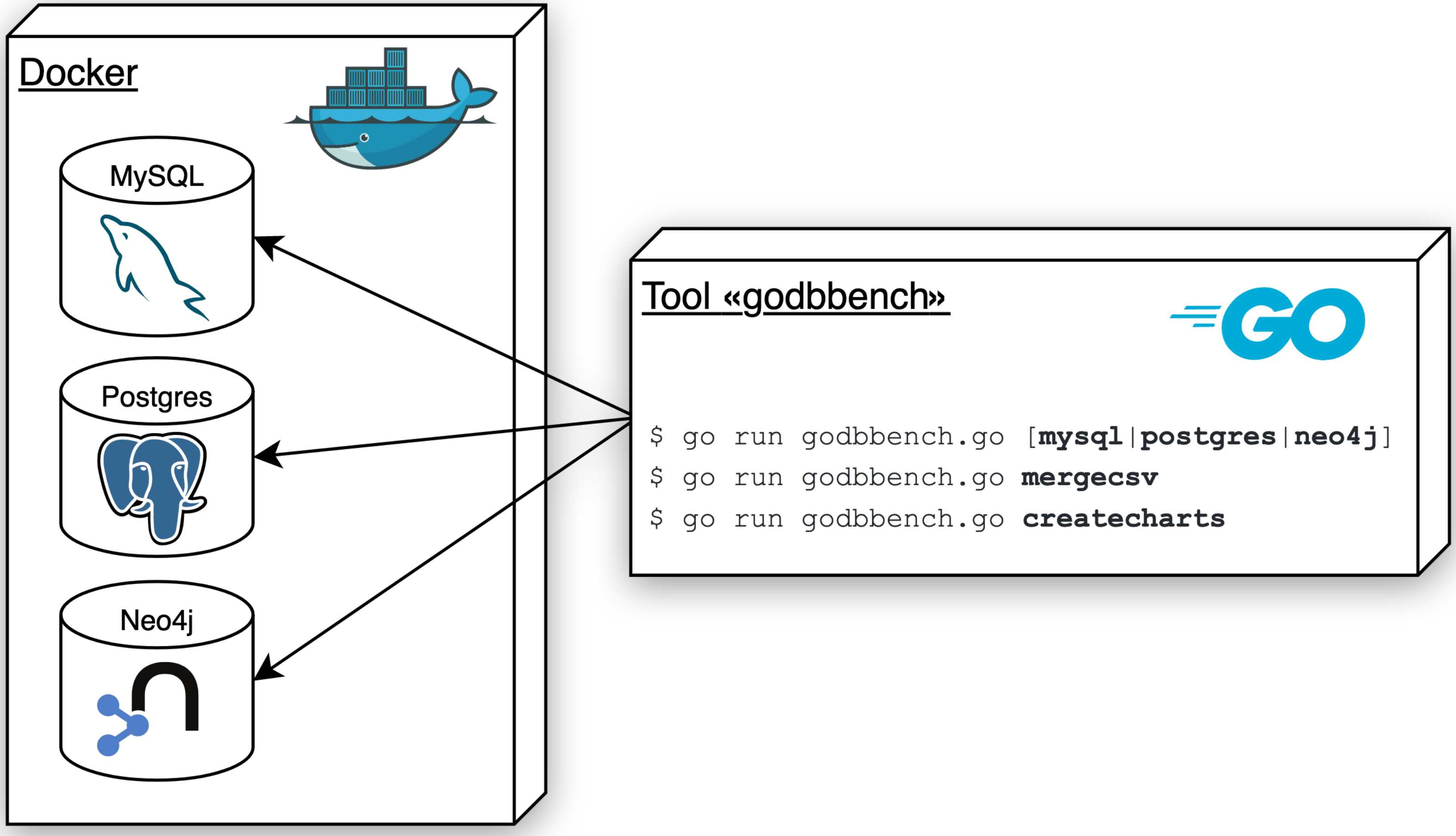
```
-- SQL
SELECT * FROM Customer c WHERE c.Age >= 18

-- Cyper
MATCH (c:Customer) WHERE c.Age > 18 RETURN c;
```

Show top clients based on revenue

```
-- SQL
SELECT c.CustomerId, c.Name, SUM(p.Total) FROM Customer c
INNER JOIN Purchase p on c.CustomerId = p.CustomerId
GROUP BY c.CustomerId, c.Name ORDER BY SUM(p.Total) DESC

-- Cyper
MATCH (c:Customer)-[:MAKES]->(p:Purchase)
RETURN c.Name, SUM(p.Total) AS TotalOrderValue ORDER BY TotalOrderValue DESC
```

System Setup

- Requirements:
  - Docker
  - Go
  - godbbench



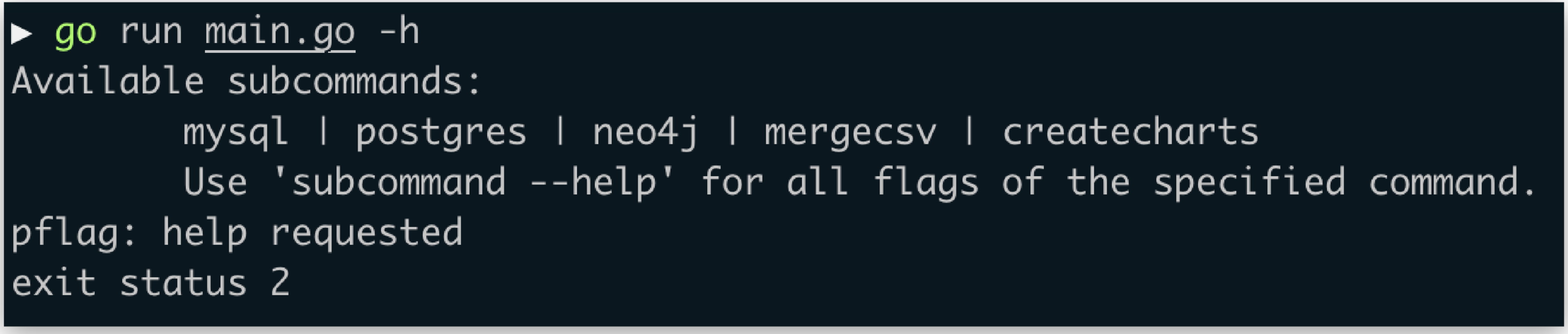
```
$ go run godbbench.go [mysql|postgres|neo4j]
$ go run godbbench.go mergecsv
$ go run godbbench.go createcharts
```

Command Line Interface (CLI)

- Open terminal and navigate to the location of `godbbench.go`

  `$ cd ~/path/to/godbbench/cmd`

- Interact with `go run godbbench.go` to see flags



```
▶ go run main.go -h
Available subcommands:
        mysql | postgres | neo4j | mergecsv | createcharts
        Use 'subcommand --help' for all flags of the specified command.
pflag: help requested
exit status 2
```

Possilbe CLI Commands

```
# run synthetic INSERT and SELECT statements against MySQL, each 100x
$ go run godbbench.go mysql --host 127.0.0.1 --port 3306 --user "root" \
    --pass "password" --iter 100 --run "inserts selects"
```

```
# run statemets of custom script against Postgres, save results in file
$ go run godbbench.go postgres --host 127.0.0.1 --port 5432 --user "postgres" \
    --pass "password" --iter 100 --script "./path/to/mysql.sql" \
    --writecsv "./path/to/results/mysql.csv"
```

```
# merge serveral result files
$ go run godbbench.go mergecsv \
    --rootDir "~/path/with/csv-files/to-be-merged"
    --targetFile "~/anypath/allresults.csv"
```

```
# visualize the benchmarking results
$ go run godbbench.go createcharts \
    --dataFile "~/anypath/allresults.csv" --charttype "line"
```

Statement Substitutions

```
INSERT INTO Customer (Id, Name, Birthday)
VALUES ( {{.Iter}}, '{{call .RandString 3 10 }}', '{{call .RandDate }}');
```

Following expressions will be substituted before the statement is executed:

`{{.Iter}}` --> The iteration counter. Will return 1 when `\benchmark once`.

`{{call .RandIntBetween 1 100}}` --> Random integer between `1` and `100`.

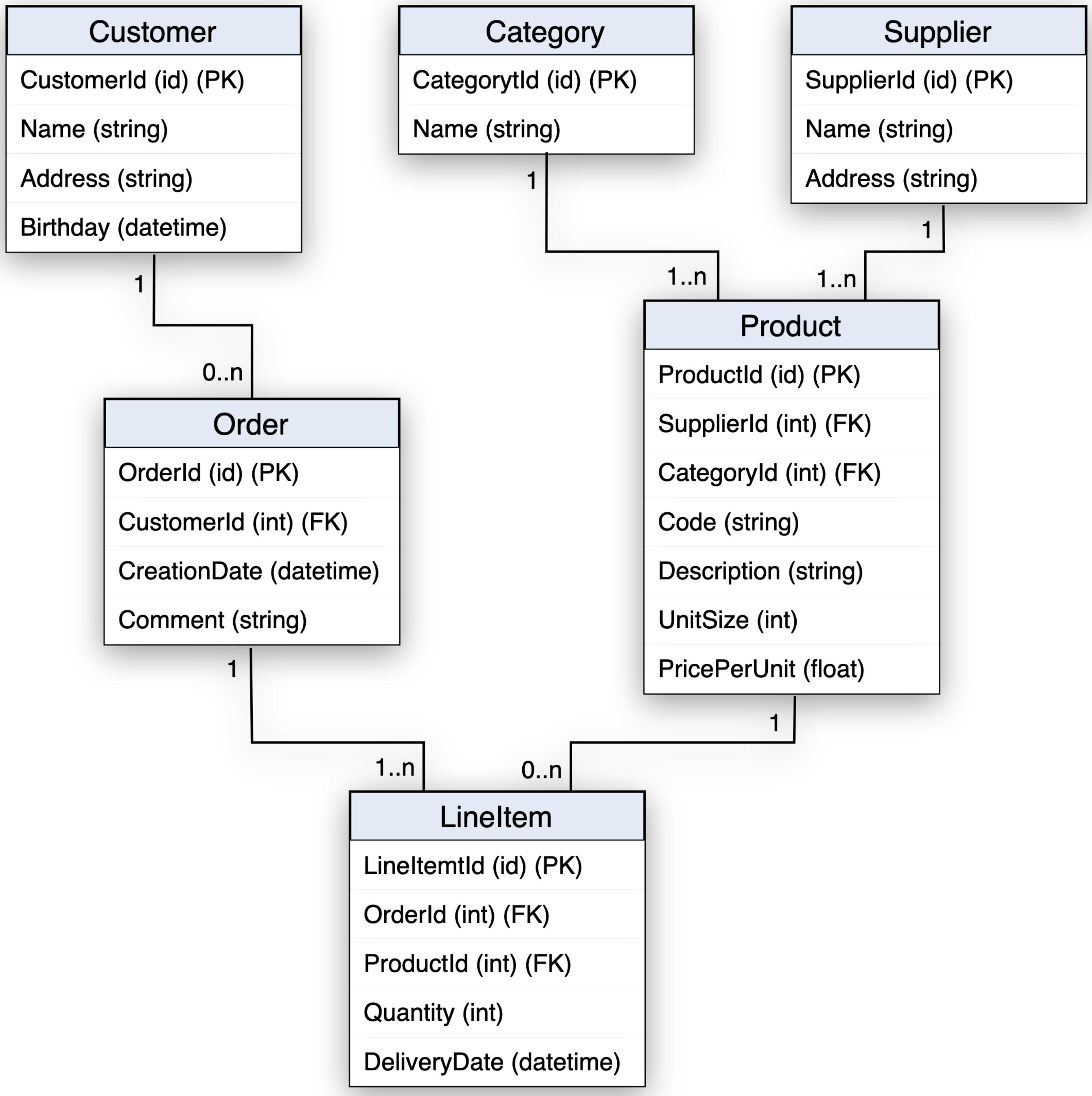
`{{call .RandFloatBetween 0 1}}` --> Random float between `0` and `1`.

`{{call .RandString 3 15}}` --> Random string with length between `3` and `15`.

`{{call .RandDate}}` --> Random date.

Custom Script (`merchant`)

```
-- INIT (illustration purposes)
\benchmark once \name initialize
DROP SCHEMA IF EXISTS godbbench CASCADE; CREATE SCHEMA godbbench;
CREATE TABLE godbbench.order (OrderId INT PRIMARY KEY, CustomerId INT NOT NULL, ... );

-- INSERTS (illustration purposes)
\benchmark loop 1.0 \name inserts
INSERT INTO godbbench.Order (OrderId, CustomerId, CreationDate, Comment)
VALUES( {{.Iter}}, (SELECT CustomerId FROM godbbench.Customer ORDER BY RANDOM() LIMIT 1),
        '{{call .RandDate }}', '{{call .RandString 0 50 }}');

-- SELECTS
\benchmark loop 1.0 \name select_simple
SELECT * FROM godbbench.Customer WHERE CustomerId = {{.Iter}}

\benchmark loop 1.0 \name select_medium
SELECT * FROM godbbench.Product p JOIN godbbench.Supplier s ON ...

\benchmark loop 1.0 \name select_complex
SELECT c.CustomerId, c.Name, SUM(li.Quantity * p.UnitSize * p.PricePerUnit) as  ...

-- CLEAN (illustration purposes)
\benchmark once \name clean
DROP SCHEMA IF EXISTS godbbench CASCADE;
```



Customer

- CustomerId (id) (PK)
- Name (string)
- Address (string)
- Birthday (datetime)

Category

- CategorytId (id) (PK)
- Name (string)

Supplier

- SupplierId (id) (PK)
- Name (string)
- Address (string)

Order

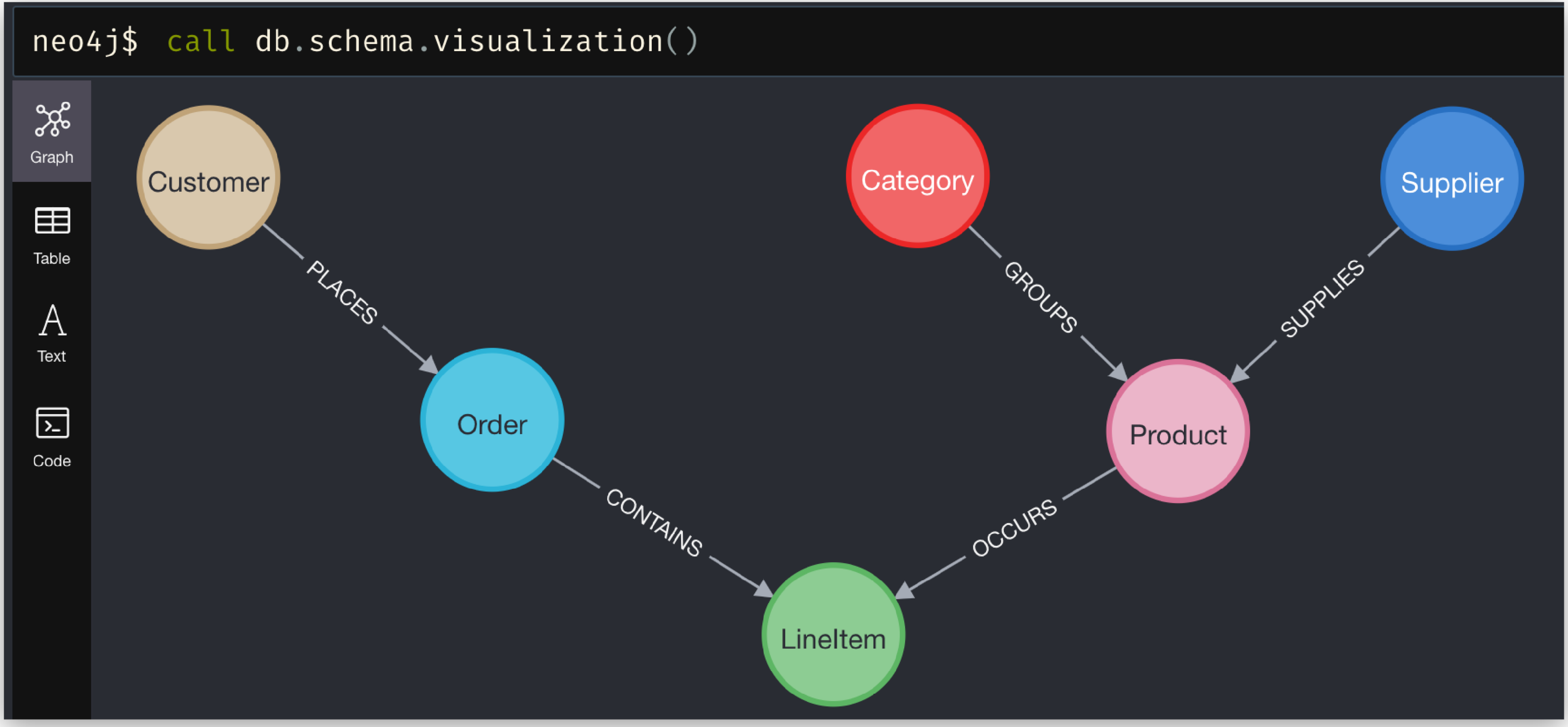
- OrderId (id) (PK)
- CustomerId (int) (FK)
- CreationDate (datetime)
- Comment (string)

Product

- ProductId (id) (PK)
- SupplierId (int) (FK)
- CategoryId (int) (FK)
- Code (string)
- Description (string)
- UnitSize (int)
- PricePerUnit (float)

LineItem

- LineItemtId (id) (PK)
- OrderId (int) (FK)
- ProductId (int) (FK)
- Quantity (int)
- DeliveryDate (datetime)



```
neo4j$ call db.schema.visualization()
```

Attention:

Relational data schemas should not directly be mapped into a graph-world.

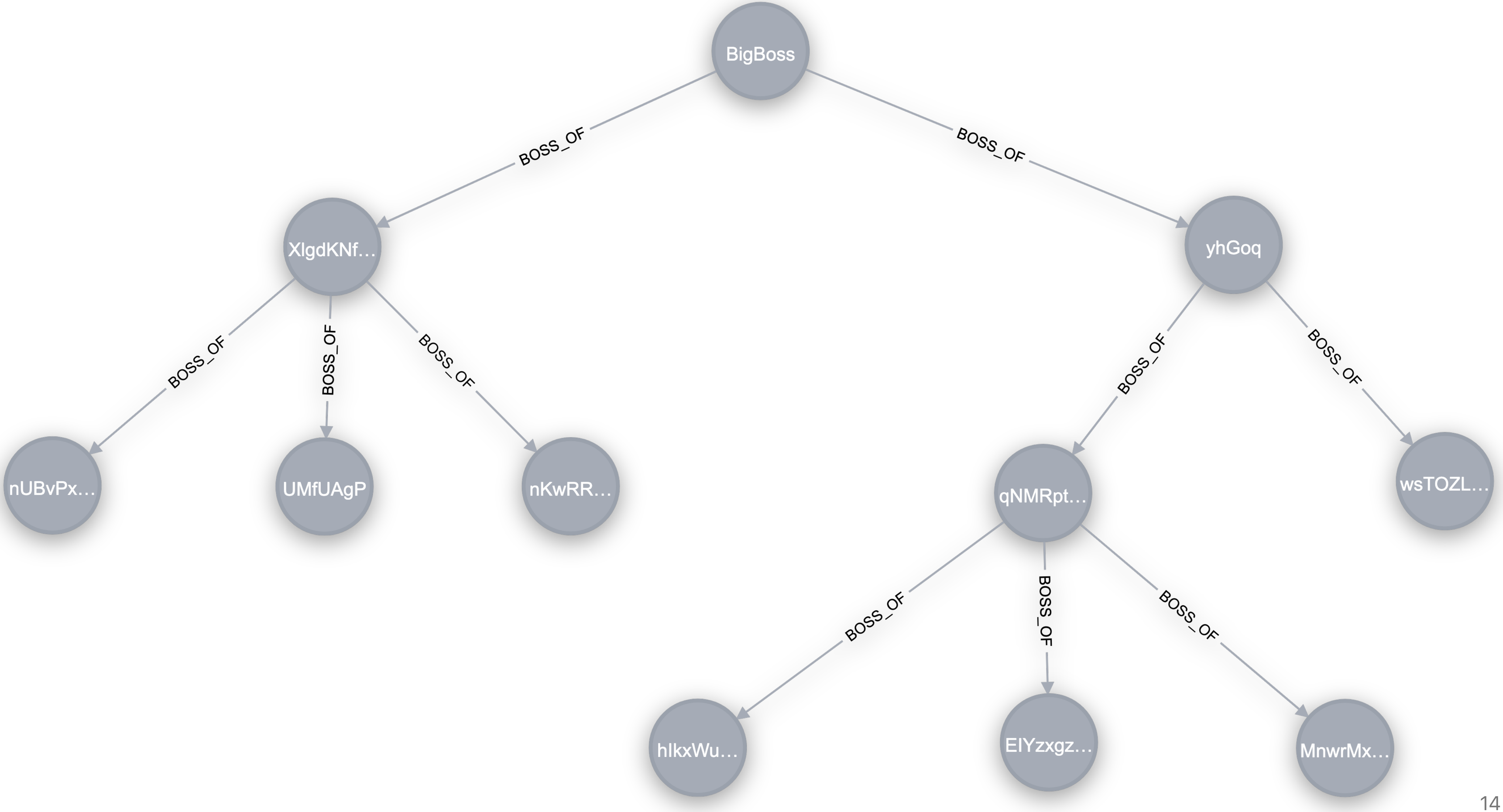
Relationships in graph-based DBs are first-class citizen that can hold information by itself.

Custom Script (`employees`)

Show all subordinates of an employee (tree queries)

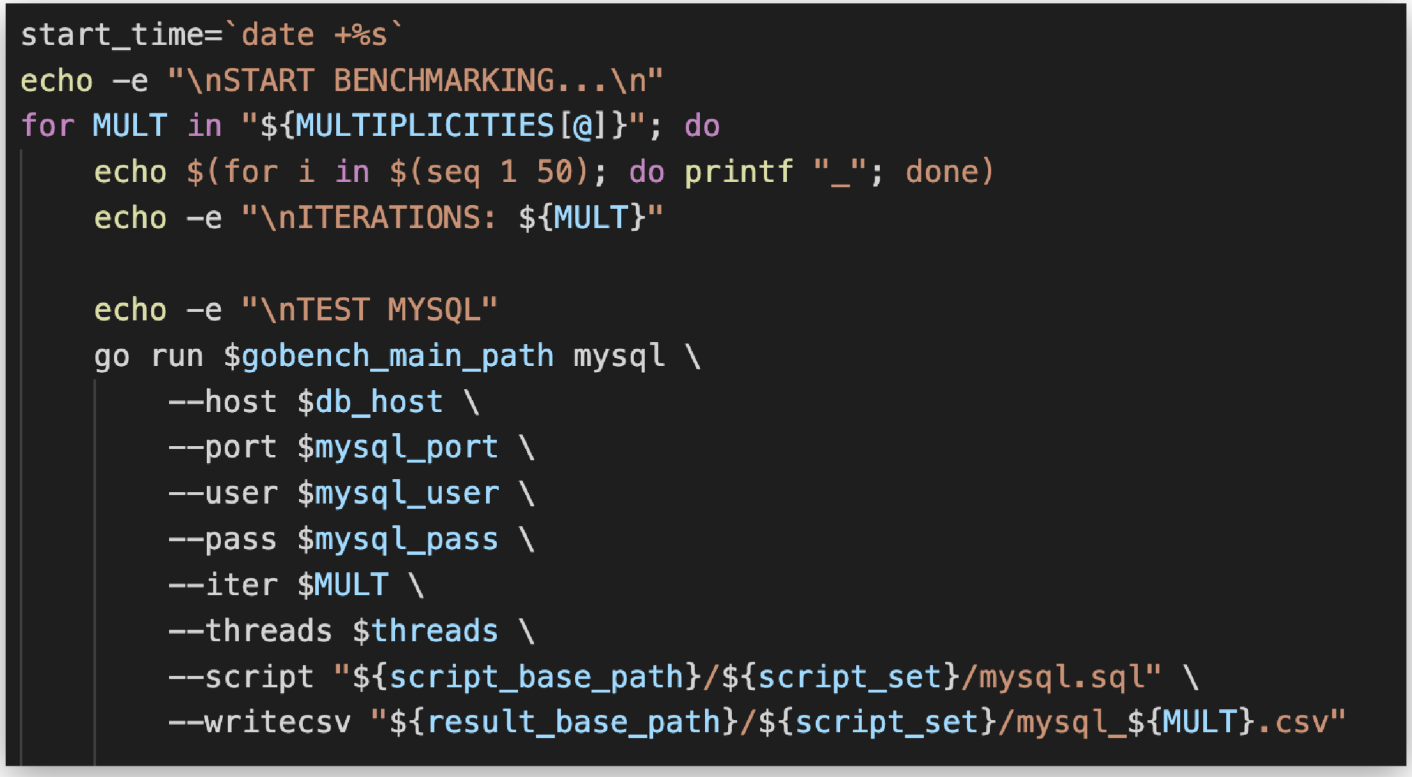
```
-- use WITH RECURISON notation in Postgres (similar in MySQL)
WITH RECURSIVE hierarchy AS (
    SELECT employeeId, firstname, boss_id, 0 AS level
    FROM employee
    WHERE employeeId = {{.Iter}}
  UNION ALL
    SELECT e.employeeId, e.firstname, e.boss_id, hierarchy.level + 1 AS level
    FROM employee e JOIN hierarchy ON e.boss_id = hierarchy.employeeId
) SELECT * FROM hierarchy;
INSERT INTO employee (firstname, boss_id, salary) VALUES ('BigBoss', null, 999999);

-- simpler query using Cypher
MATCH (boss)-[:BOSS_OF*1..]->(sub) WHERE boss.employeeId={{.Iter}} RETURN sub;
```

> see example graph on next slide ...


BigBoss
BOSS_OF
BOSS_OF
XlgdKNf…
yhGoq
BOSS_OF
BOSS_OF
BOSS_OF
BOSS_OF
BOSS_OF
nUBvPx…
UMfUAgP
nKwRR…
qNMRpt…
wsTOZL…
BOSS_OF
BOSS_OF
BOSS_OF
hIkxWu…
EIYzxgz…
MnwrMx…

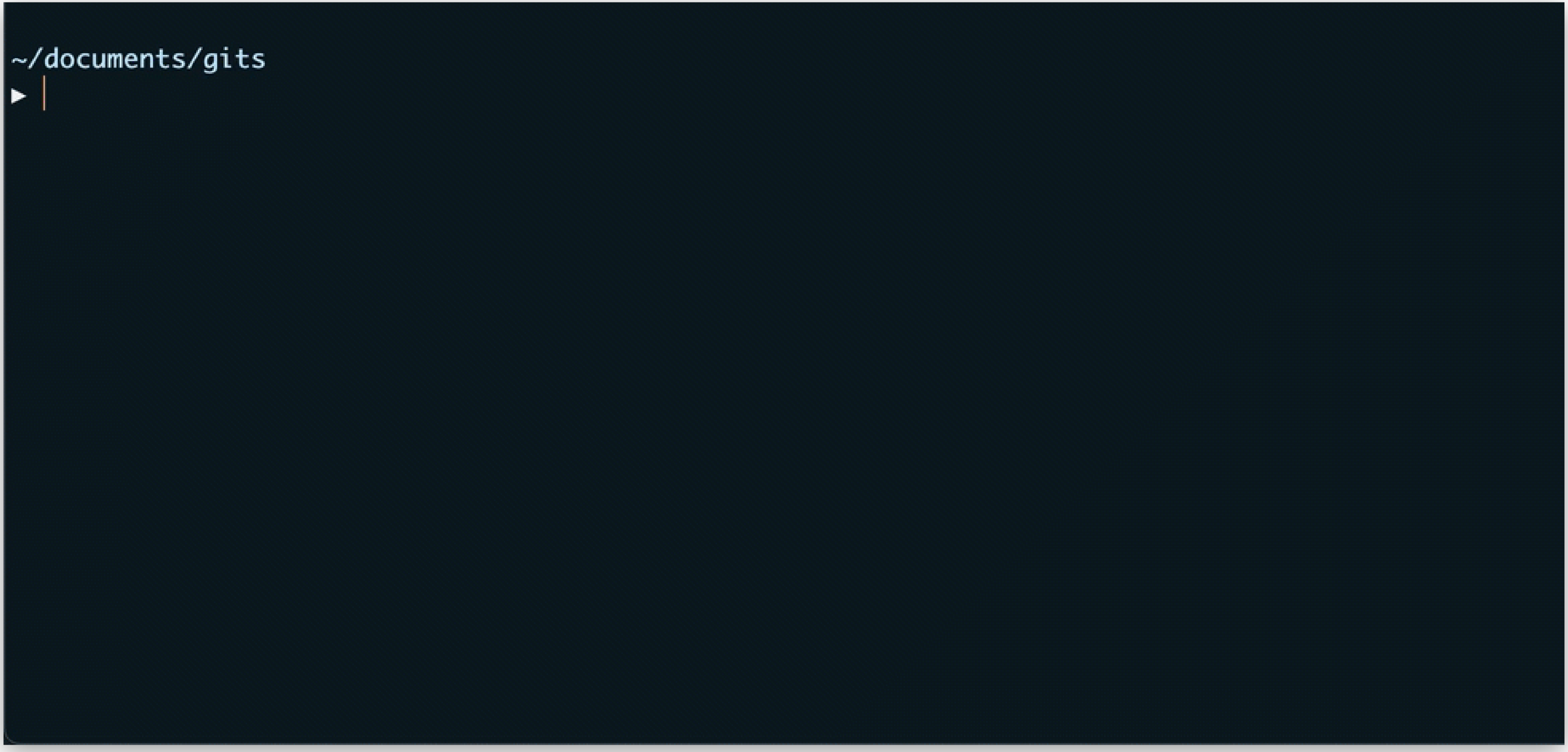
Automation

```
$ bash bashscript.sh
```



```
start_time=`date +%s`
echo -e "\nSTART BENCHMARKING...\n"
for MULT in "${MULTIPLICITIES[@]}"; do
    echo $(for i in $(seq 1 50); do printf "_"; done)
    echo -e "\nITERATIONS: ${MULT}"

    echo -e "\nTEST MYSQL"
    go run $gobench_main_path mysql \
        --host $db_host \
        --port $mysql_port \
        --user $mysql_user \
        --pass $mysql_pass \
        --iter $MULT \
        --threads $threads \
        --script "${script_base_path}/${script_set}/mysql.sql" \
        --writecsv "${result_base_path}/${script_set}/mysql_${MULT}.csv"
```

> see demo on next slide...

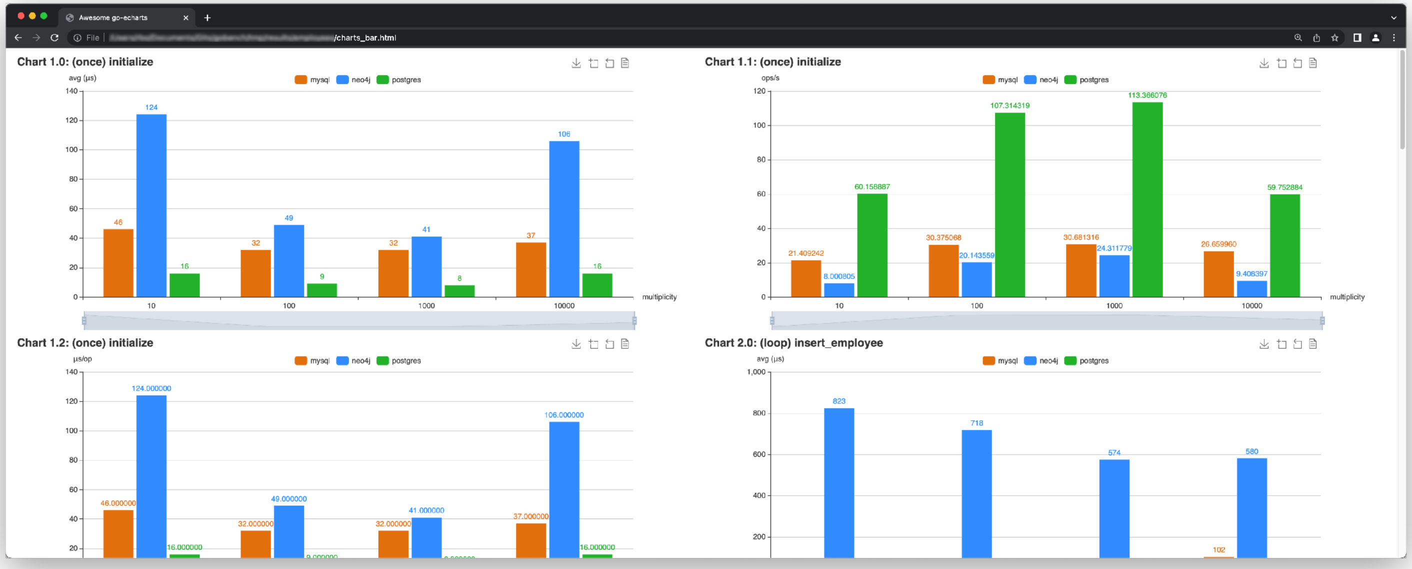


```
~/documents/gits
▶ 
```

Result Analysis

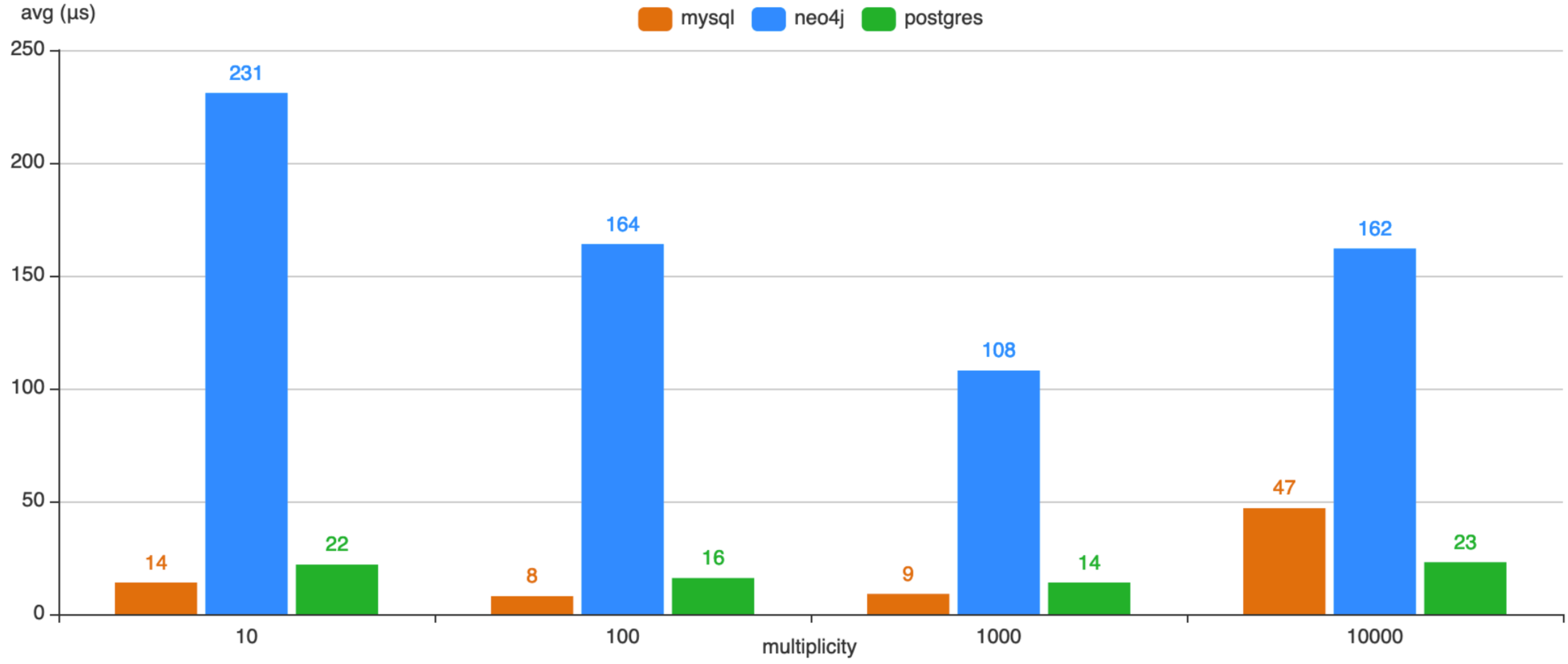
Generating a `chart.html` file to visualize

- average amount of microseconds (`µs`) per benchmark (the lower the better)
- operations per second (the higher the better)
- microseconds per operation (the lower the better)



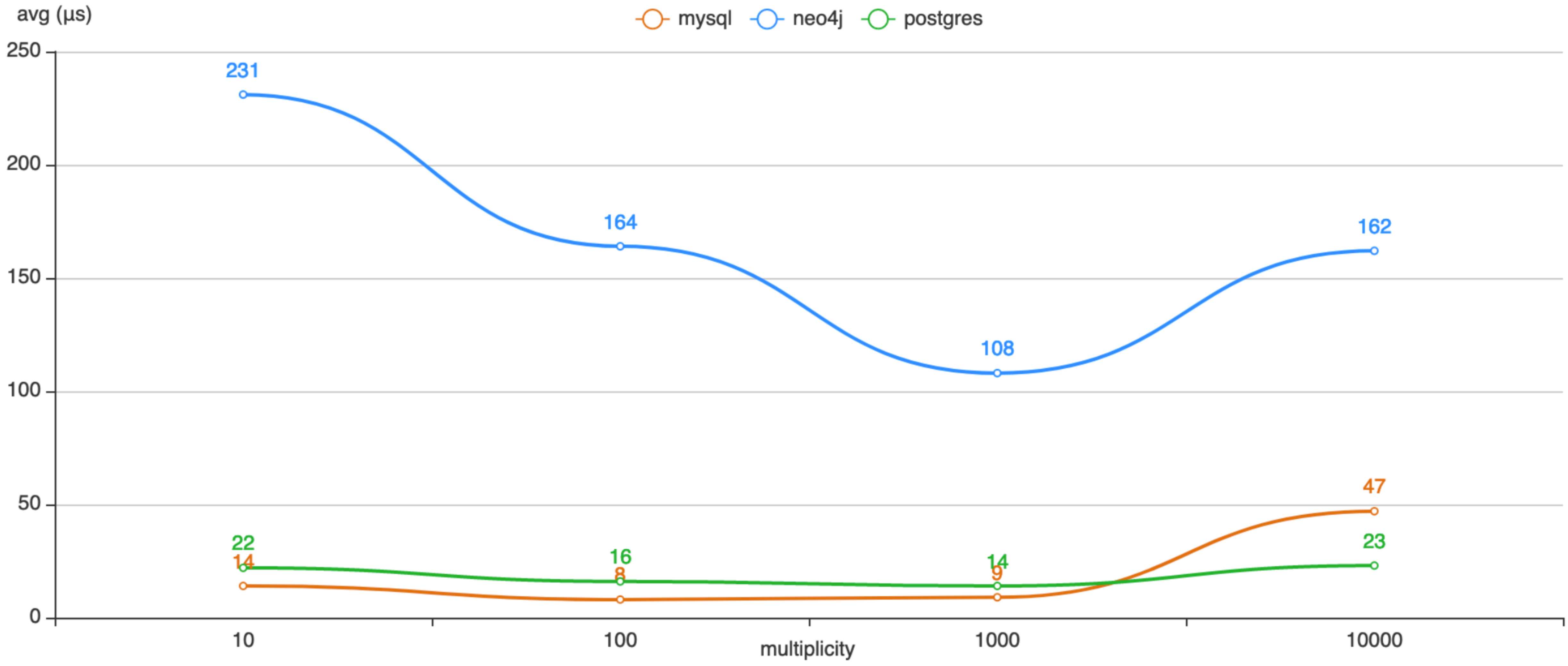
1 second (s) = 1'000'0000 microseconds (µs)

Chart 3.0: (loop) select_before_index



1 second (s) = 1'000'0000 microseconds (µs)

Chart 3.0: (loop) select_before_index



1 second (s) = 1'000'0000 microseconds (µs)

Conclusion & Future Work

- todo

References

- Bechberger, D., & Perryman, J. (2020). Graph databases in Action: Examples in Gremlin. Manning.
- Bush, J. (2020). Learn SQL Database Programming: Query and manipulate databases from popular relational database servers using SQL.
- Chauhan, C., & Kumar, D. (2017). PostgreSQL High Performance Cookbook: Mastering query optimization, database monitoring, and performance-tuning for PostgreSQL. Packt Publishing.
- Codd, E. F. (2002). A Relational Model of Data for Large Shared Data Banks. In M. Broy & E. Denert (Eds.), Software Pioneers (pp. 263–294). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-59412-0_16
- Elmasri, R., & Navathe, S. (2011). Fundamentals of Database Systems (6th ed). Addison-Wesley.
- Fleming, P. J., & Wallace, J. J. (1986). How not to lie with statistics: The correct way to summarize benchmark results. Communications of the ACM, 29(3), 218–221. https://doi.org/10.1145/5666.5673
- Gray, J. (Ed.). (1994). The Benchmark Handbook for Database and Transaction Processing Systems (2. ed., 2. [print.]). Morgan Kaufmann.
- Gregg, B. (2020). Systems Performance: Enterprise and the Cloud (Second). Addison-Wesley.
- Meier, A., & Kaufmann, M. (2019). SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management. Springer Vieweg.
- Needham, M., & Hodler, A. E. (2019). Graph Algorithms: Practical Examples in Apache Spark and Neo4j (First edition). O'Reilly Media.
- Peixoto, T. P. (n.d.). What is graph-tool? Graph-Tool. Retrieved 20 March 2022, from https://graph-tool.skewed.de/
- Robinson, I., Webber, J., & Eifrem, E. (2015). Graph Databases: New Opportunities for Connected Data.
- Scalzo, B. (2018). Database Benchmarking and Stress Testing: An Evidence-Based Approach to Decisions on Architecture and Technology. Springer Science+Business Media, LLC.
- Stopford, B. (2012, August 17). Thinking in Graphs: Neo4J. http://www.benstopford.com/2012/08/17/thinking-in-graphs-neo4j/

Thanks