Performance Comparison of select and construct queries of triplestores

-

on the example of the JVMG project

Tobias Malmsheimer Stuttgart Media University

Japanese Visual Media Graph

- Research project at Stuttgart Media University
- Goal: Build a knowledge graph on the domain of Japanese visual media (manga, anime, computer games)
- Data source: Enthusiast communities on the web

Our Workflow

- Ingestion workflow
  - Retrieve data dump / load data from API in proprietary data format
  - Assess data structure and organisation and create an OWL ontology
  - Convert into RDF triples
  - Load into Fuseki database
- Integration workflow
  - Match semantically equivalent entities into clusters
  - Merge information from all sources (create a merged entity)
  - Match semantically equivalent entity properties
  - Merge property values (reduce redundancy)

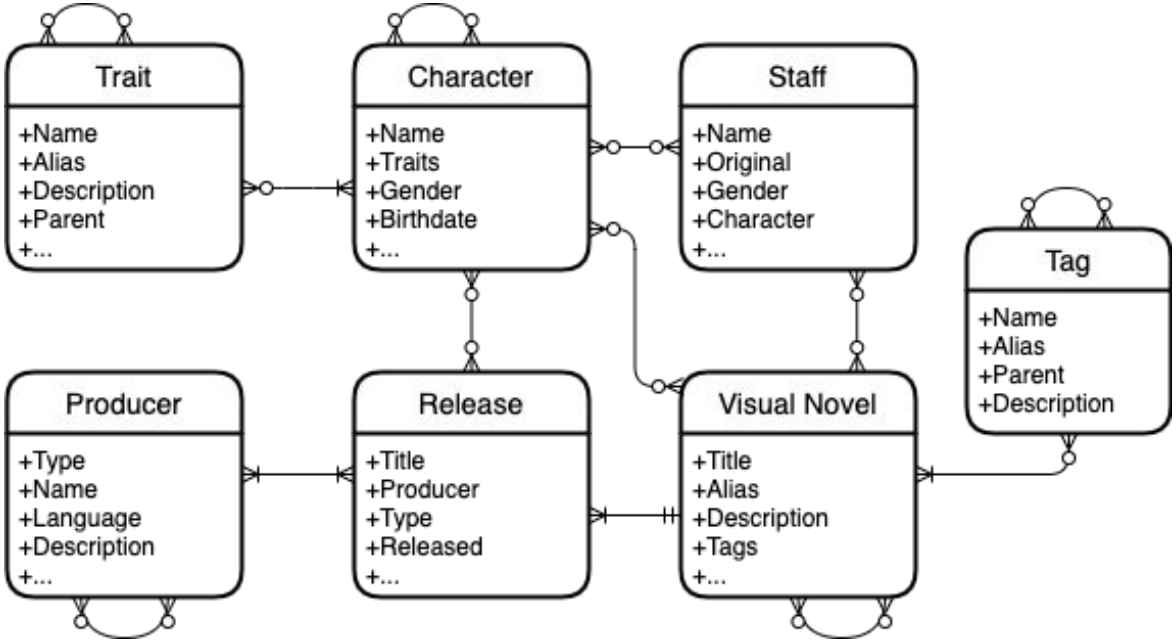
Why we need a Database

- Need to store and access our knowledge graph

- Our frontend expects a sparql-endpoint
  - User can explore our knowledge graph

- Tiny use cases (TUCs)
  - Media researcher use the knowledge graph for their research questions
  - In this way, we can improve the knowledge graph and the supporting tools

vndb abstract data model



Source: https://zenodo.org/record/5506936

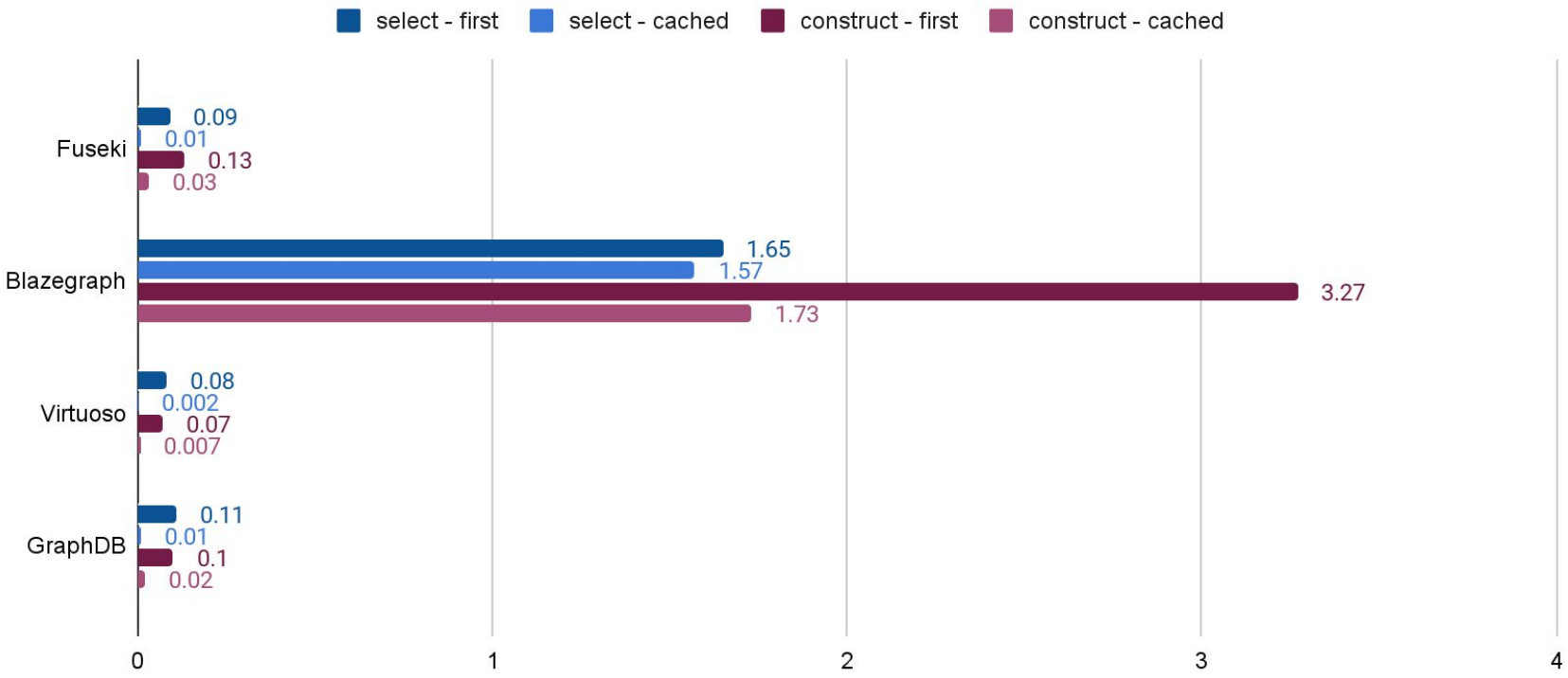
How does vndb look like

Type	Count
Character	90077
Trait	2237
Tag	2053
Producer	10394
Staff	21164
Visual Novel	28190
Visual Novel Release	71349

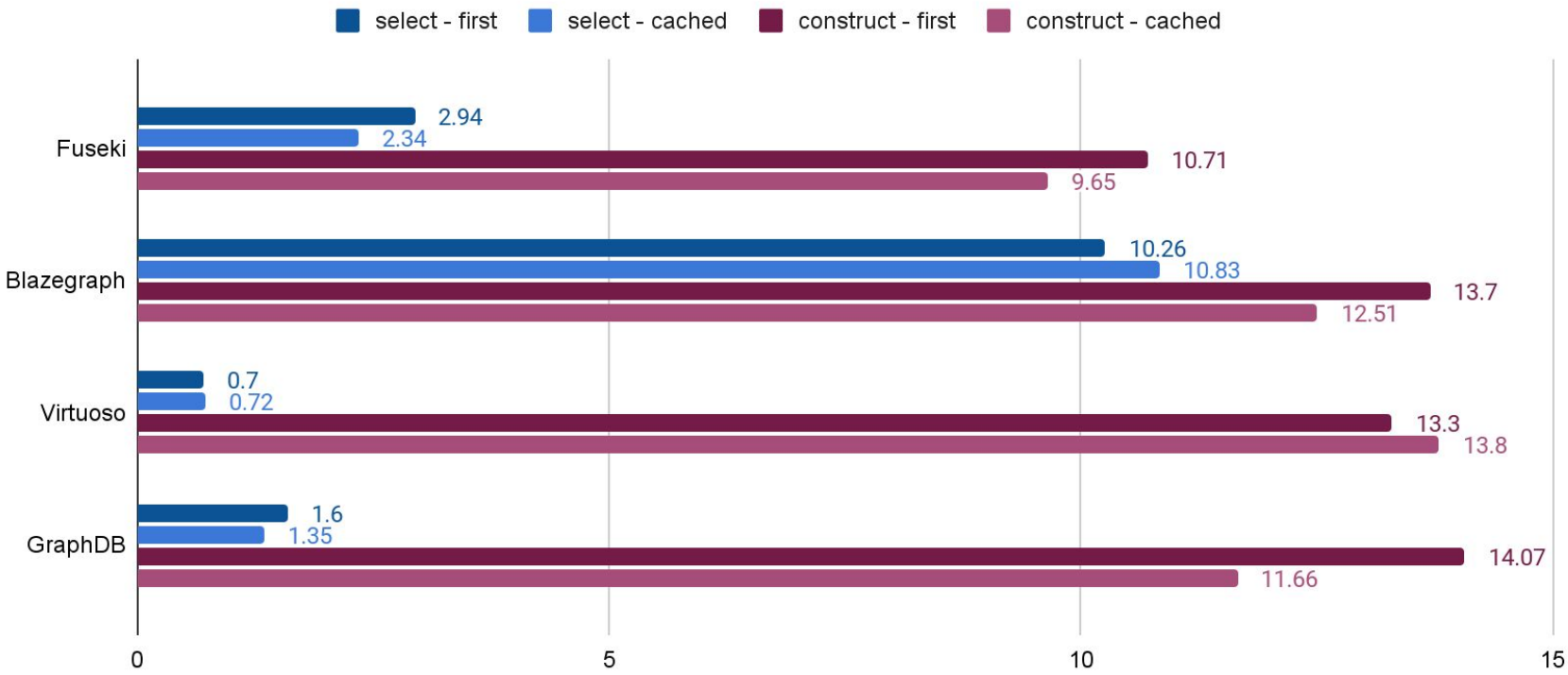
Disclaimer

- Just my opinion
- I did not optimize everything for each database
- Interested in the “out of the box” experience
- Therefore critique and bad numbers do not mean the database is bad

Small Query - First Evaluation



Medium Query - First Evaluation



Performance - Queries

- 2 different queries
  - Usual Queries we often use
- Small Query
  - Character page from Phorni
  - 96 triple
  - Small character page
- Medium Query
  - Overview of all vndb Characters
  - ~180k triple
  - Name of every entity which has type Character

Performance - CPU

- Python
  - Sparqlwrapper and rdflib for sparql endpoints
- Complete round trip
  - Executing sparql query
  - Serializing result
  - Average of 5 Queries, after warm-up queries
- Two kinds of queries
  - Select, assumption: better optimized
  - Construct
- Serialization methods
  - XML, JSON, CSV, JSONLD, Turtle, n-triples

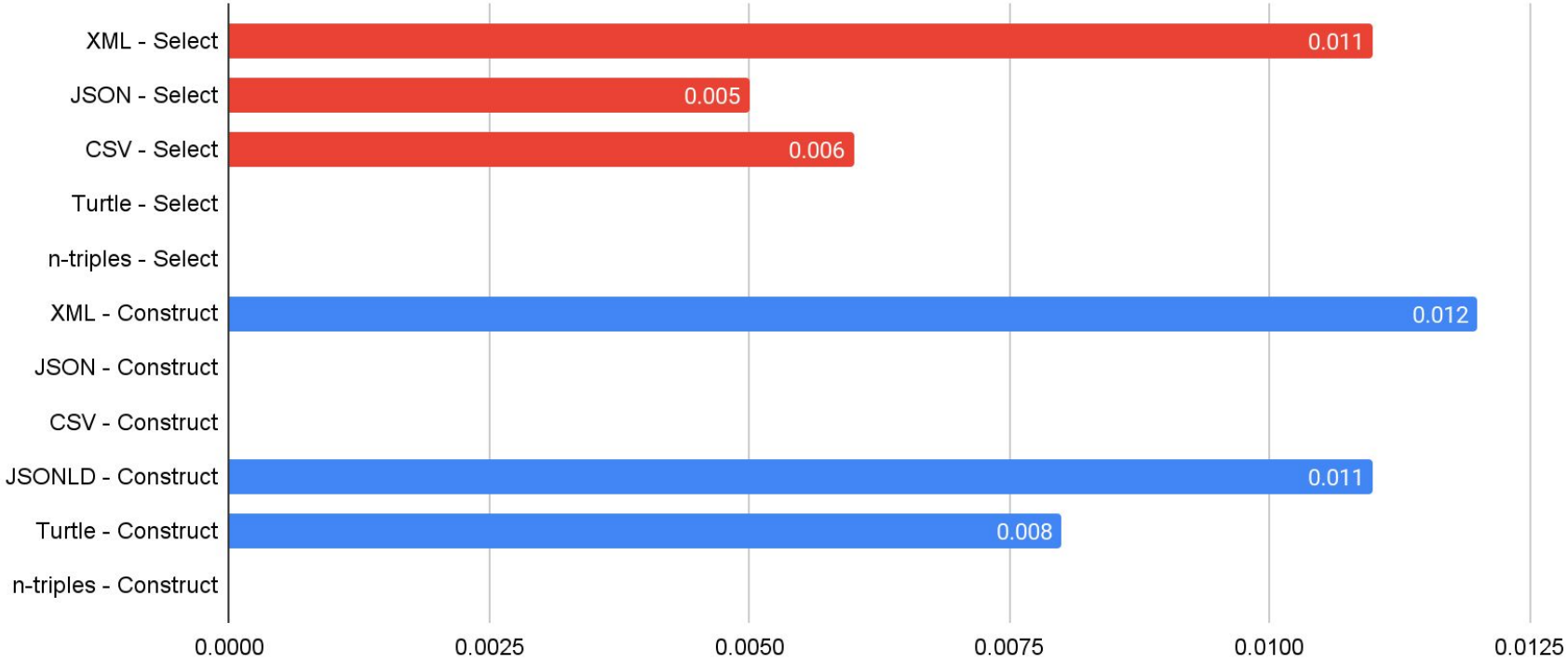
Query - Select

```
PREFIX label: <http://www.w3.org/2000/01/rdf-schema#label>
PREFIX graph_label: <http://mediagraph.link/jvmg/ont/shortLabel>
SELECT
  ?subject ?predicate ?object ?subjectLabel ?predicateLabel ?objectLabel ?graph ?graphLabel
WHERE {
  {
    GRAPH ?graph {
      ?subject ?predicate ?object . FILTER ( ?subject = <http://mediagraph.link/vndb/character/1> )
    }
    OPTIONAL { ?graph     graph_label: ?graphLabel }
    OPTIONAL { ?object    label:       ?objectLabel }
    OPTIONAL { ?predicate label:       ?predicateLabel }
  } UNION {
    GRAPH ?graph {
      ?subject ?predicate ?object . FILTER ( ?object = <http://mediagraph.link/vndb/character/1> )
    }
    OPTIONAL { ?graph     graph_label: ?graphLabel }
    OPTIONAL { ?subject   label:       ?subjectLabel }
    OPTIONAL { ?predicate label:       ?predicateLabel }
  }
}
```

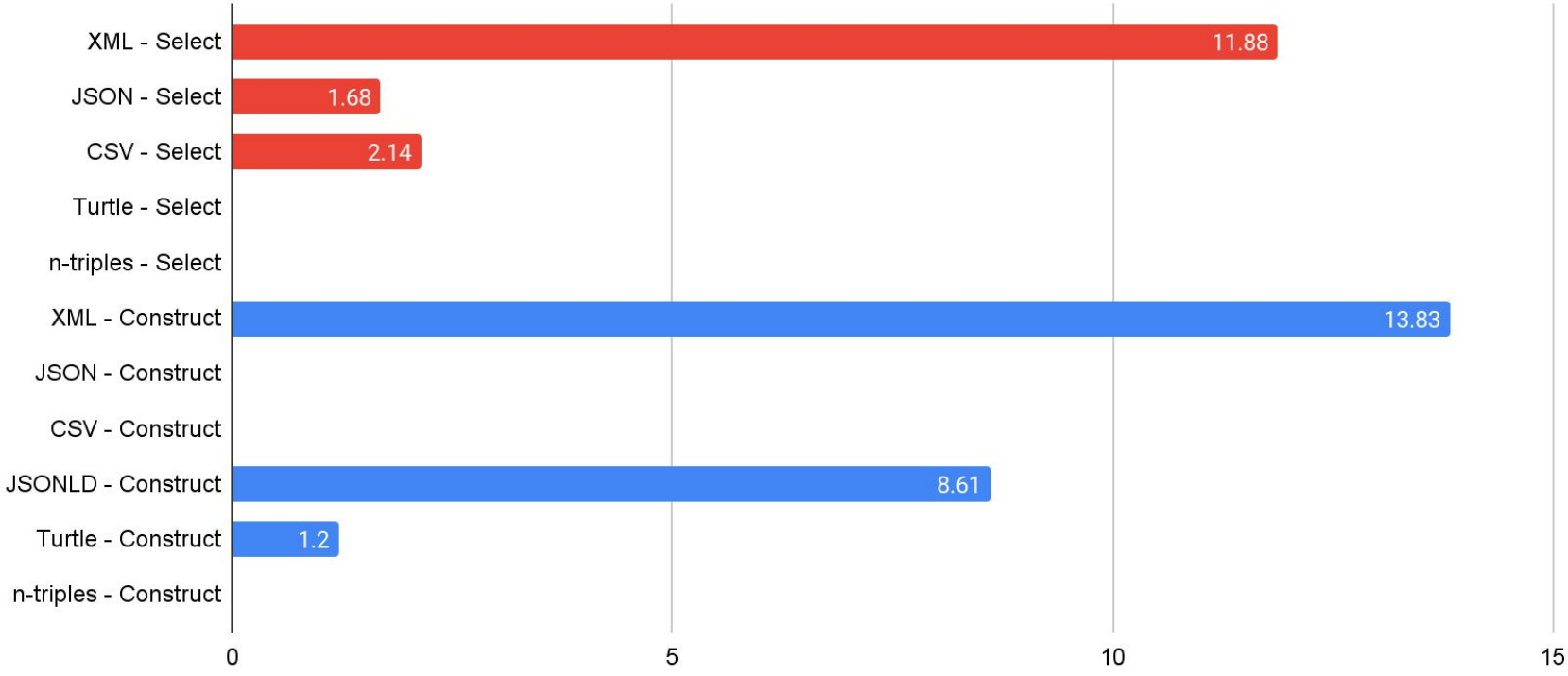
Query - Construct

```
PREFIX label: <http://www.w3.org/2000/01/rdf-schema#label>
PREFIX graph_label: <http://mediagraph.link/jvmg/ont/shortLabel>
CONSTRUCT {
  ?subject   ?predicate   ?object .
  ?graph     graph_label: ?graphLabel .
  ?object    label:       ?objectLabel .
  ?subject   label:       ?subjectLabel .
  ?predicate label:       ?predicateLabel .
} WHERE {
  {
    GRAPH ?graph {
      ?subject ?predicate ?object . FILTER ( ?subject = <http://mediagraph.link/vndb/character/1> )
    }
    OPTIONAL { ?graph     graph_label: ?graphLabel }
    OPTIONAL { ?object    label:       ?objectLabel }
    OPTIONAL { ?predicate label:       ?predicateLabel }
  } UNION {
    GRAPH ?graph {
      ?subject ?predicate ?object . FILTER ( ?object = <http://mediagraph.link/vndb/character/1> )
    }
    OPTIONAL { ?graph     graph_label: ?graphLabel }
    OPTIONAL { ?subject   label:       ?subjectLabel }
    OPTIONAL { ?predicate label:       ?predicateLabel }
  }
}
```

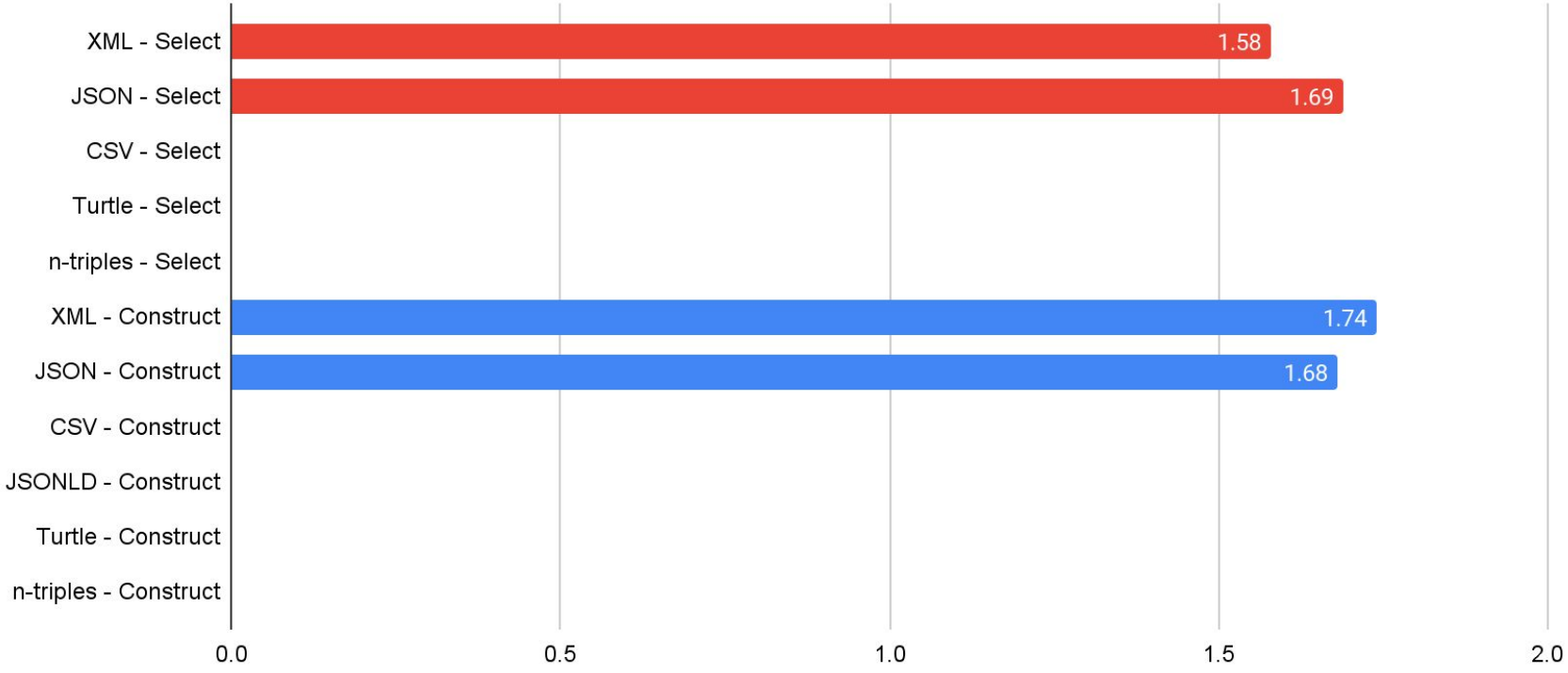
Fuseki - Small Query



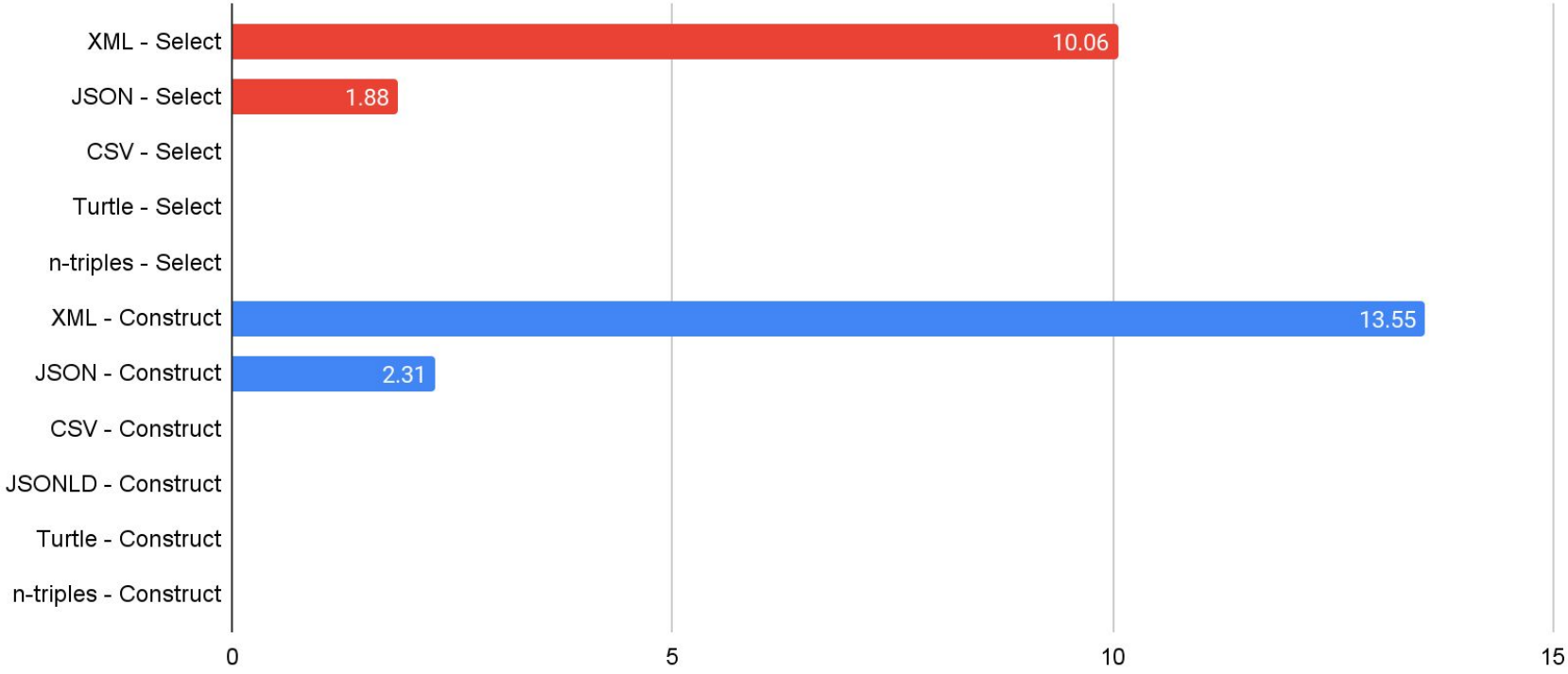
Fuseki - Medium Query



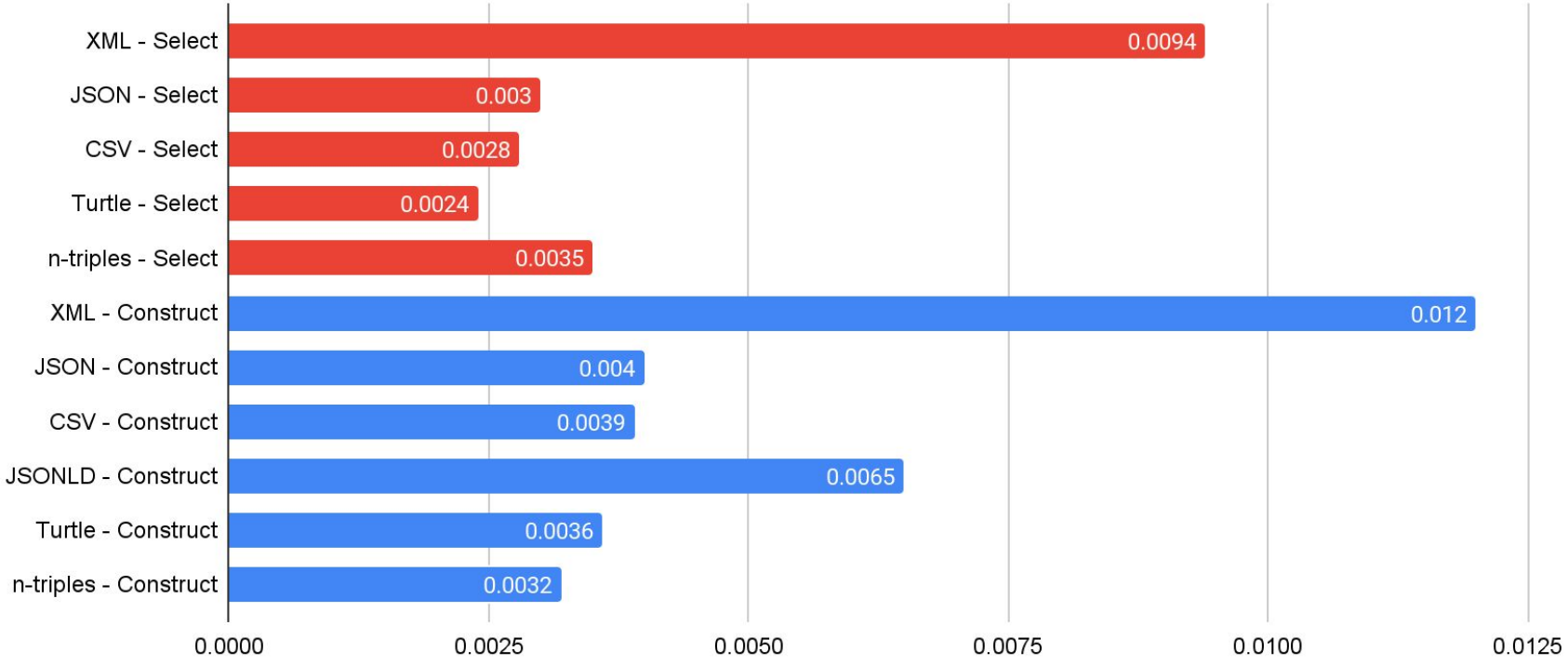
Blazegraph - Small Query



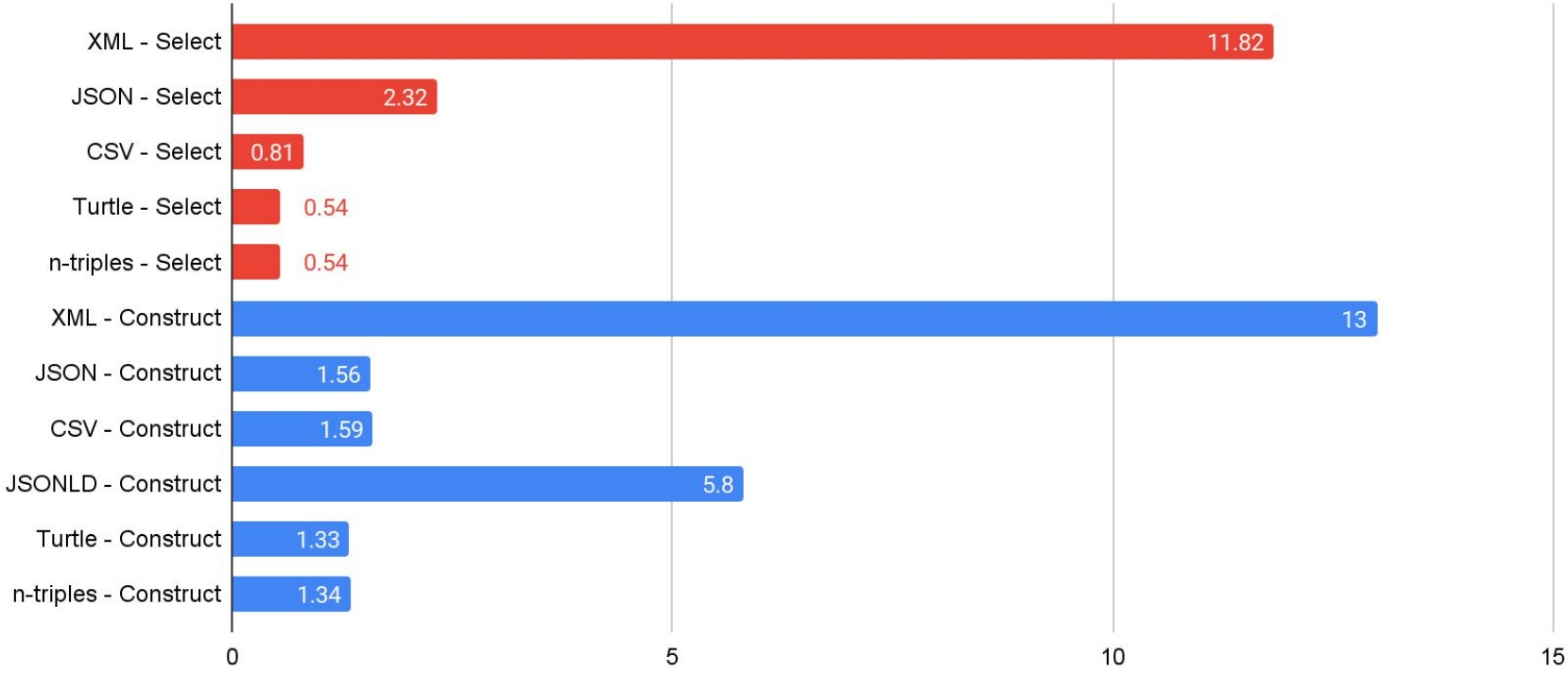
Blazegraph - Medium Query



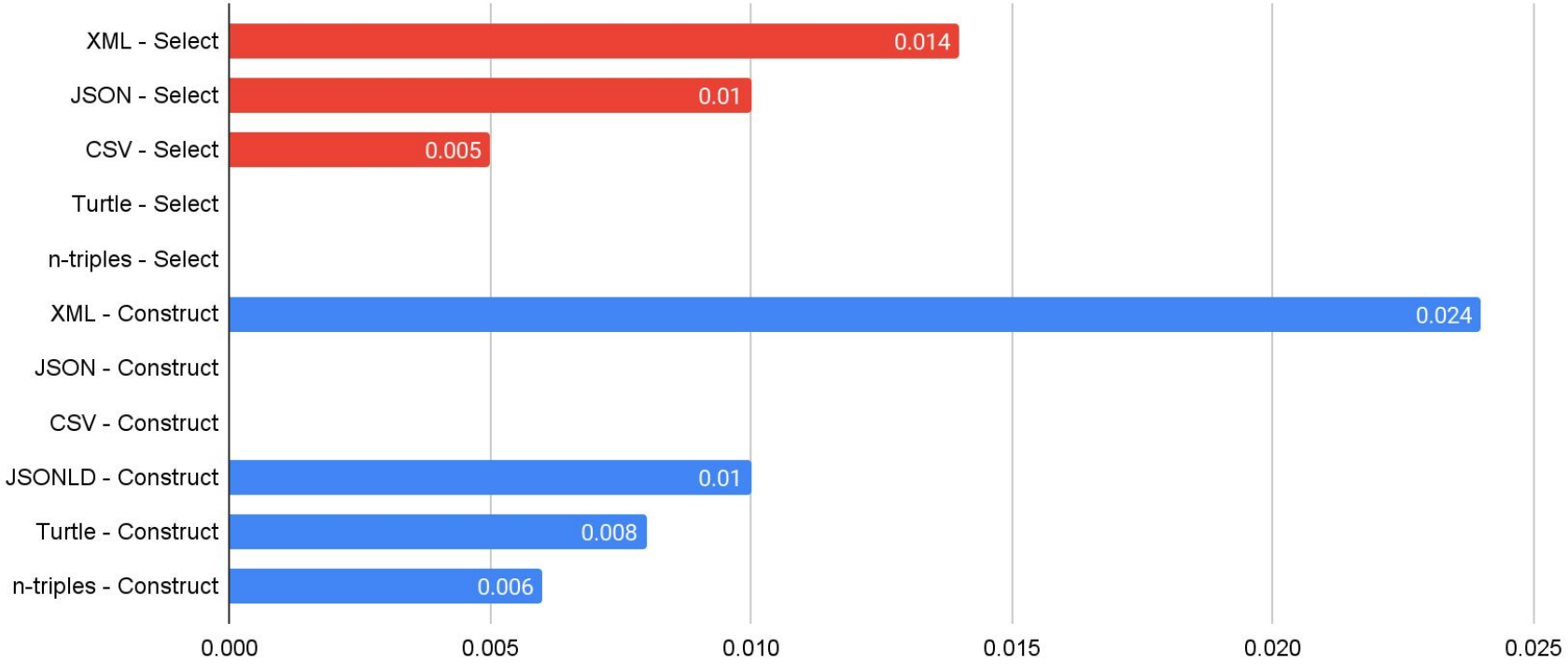
Virtuoso - Small Query



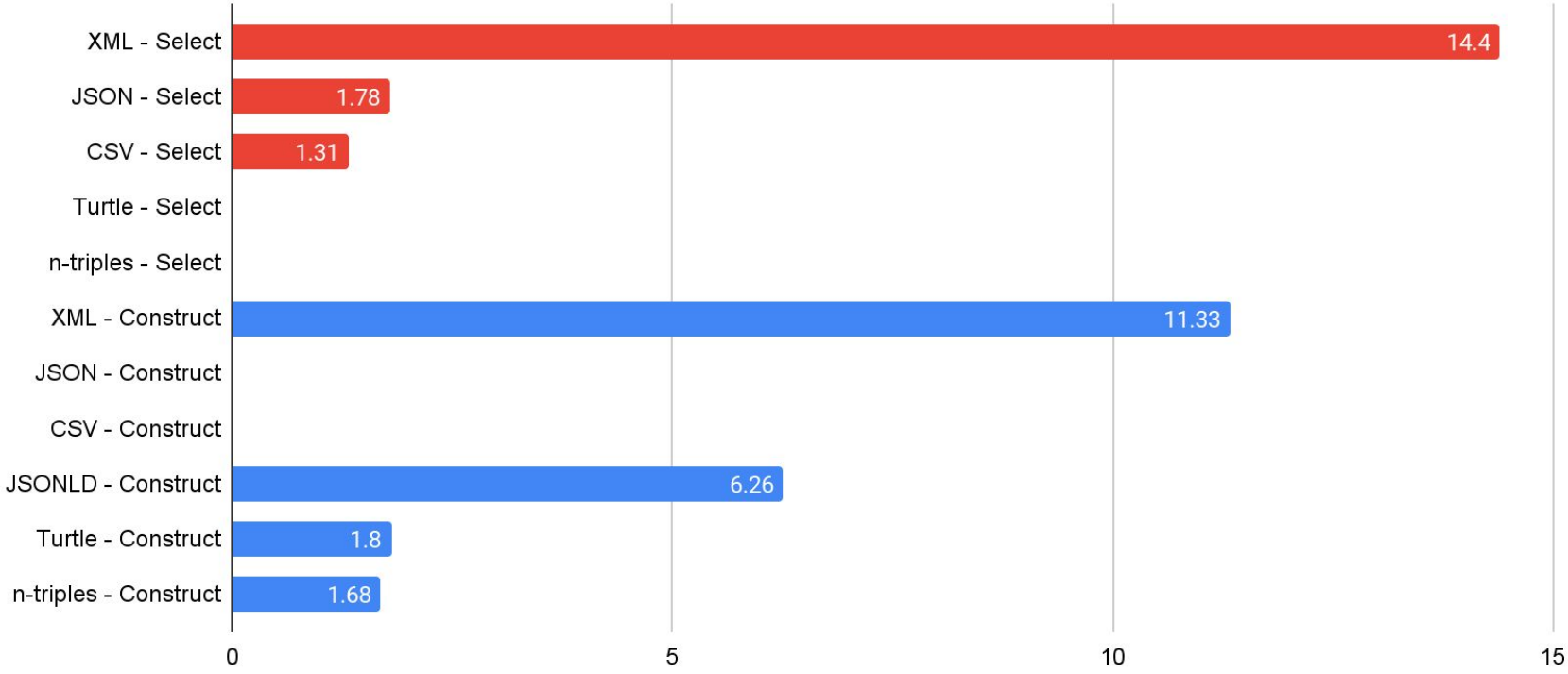
Virtuoso - Medium Query



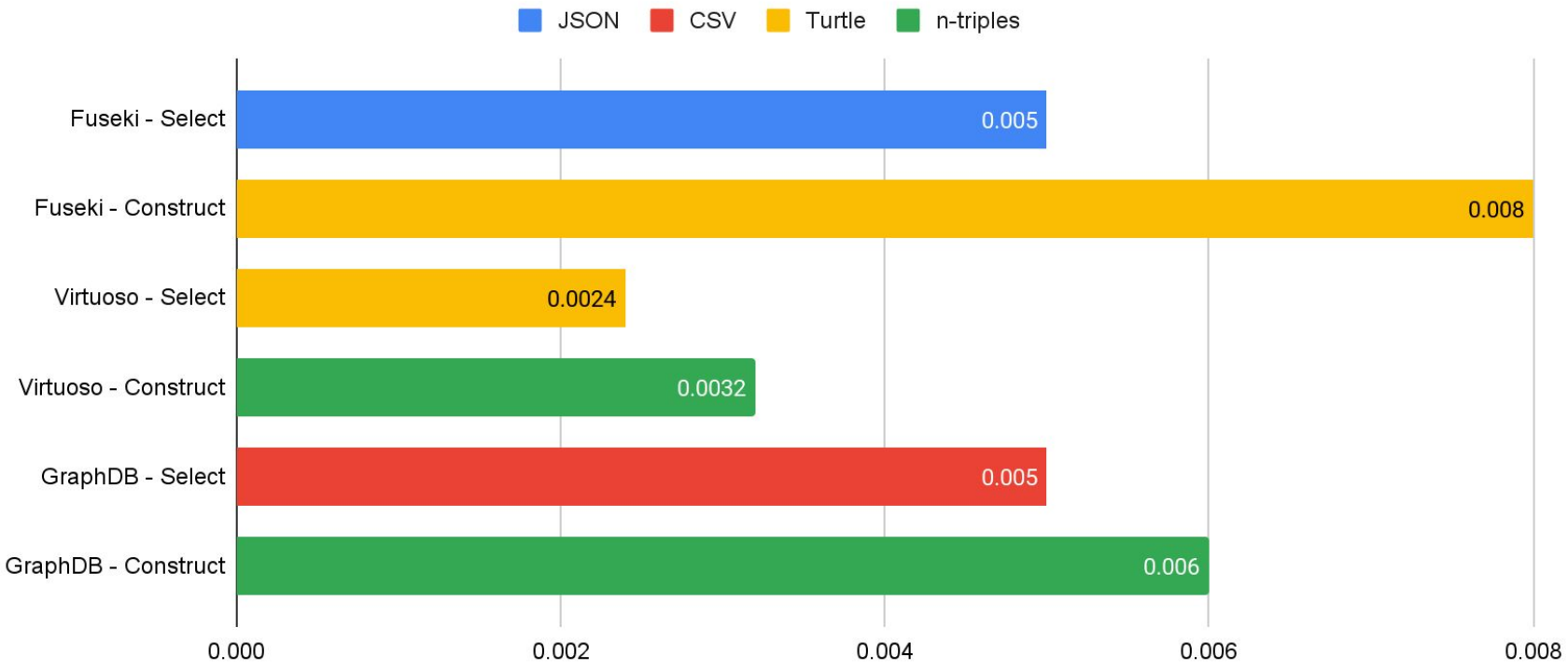
GraphDB - Small Query



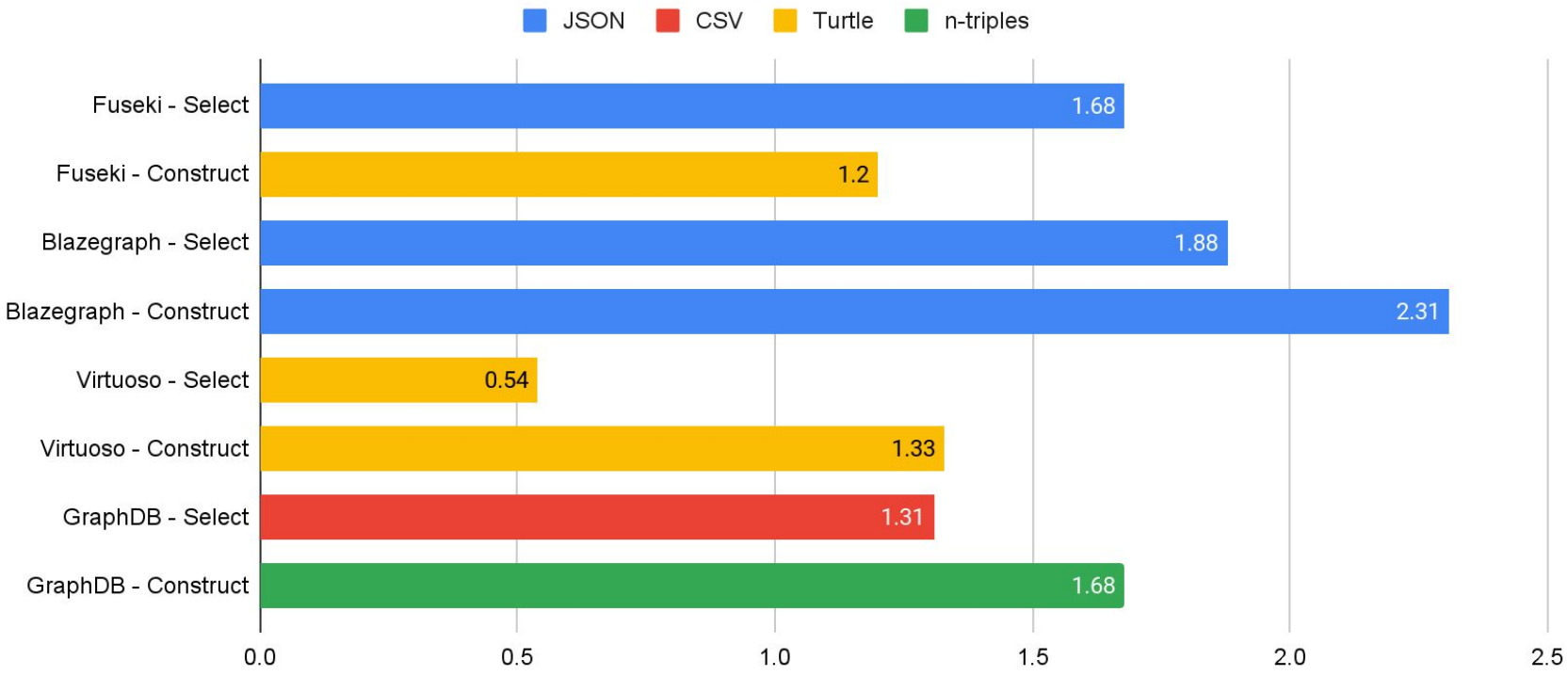
GraphDB - Medium Query



Small Query


JSON
CSV
Turtle
n-triples
Fuseki - Select
Fuseki - Construct
Virtuoso - Select
Virtuoso - Construct
GraphDB - Select
GraphDB - Construct
0.005
0.008
0.0024
0.0032
0.005
0.006
0.000
0.002
0.004
0.006
0.008

Medium Query



Conclusion

- Select queries tends to be a bit faster
- Huge performance difference when using different serialization formats
  - But: not all serialization formats allow all features (quads in fuseki)

- Your mileage may vary: test combinations of query types and serialization formats

Questions?