Complete a Short Survey

Responses for informational purposes only

Index Strategy Guide

Greg Dostatni DBA @ Command Prompt, Inc.
Postgres Conference 2025

Introduction

COMMAND PROMPT, INC.

Goals

- This is a guide for making decisions about indexes
- Everything is about trade-offs and balance
- Questions are encouraged

COMMAND PROMPT, INC.

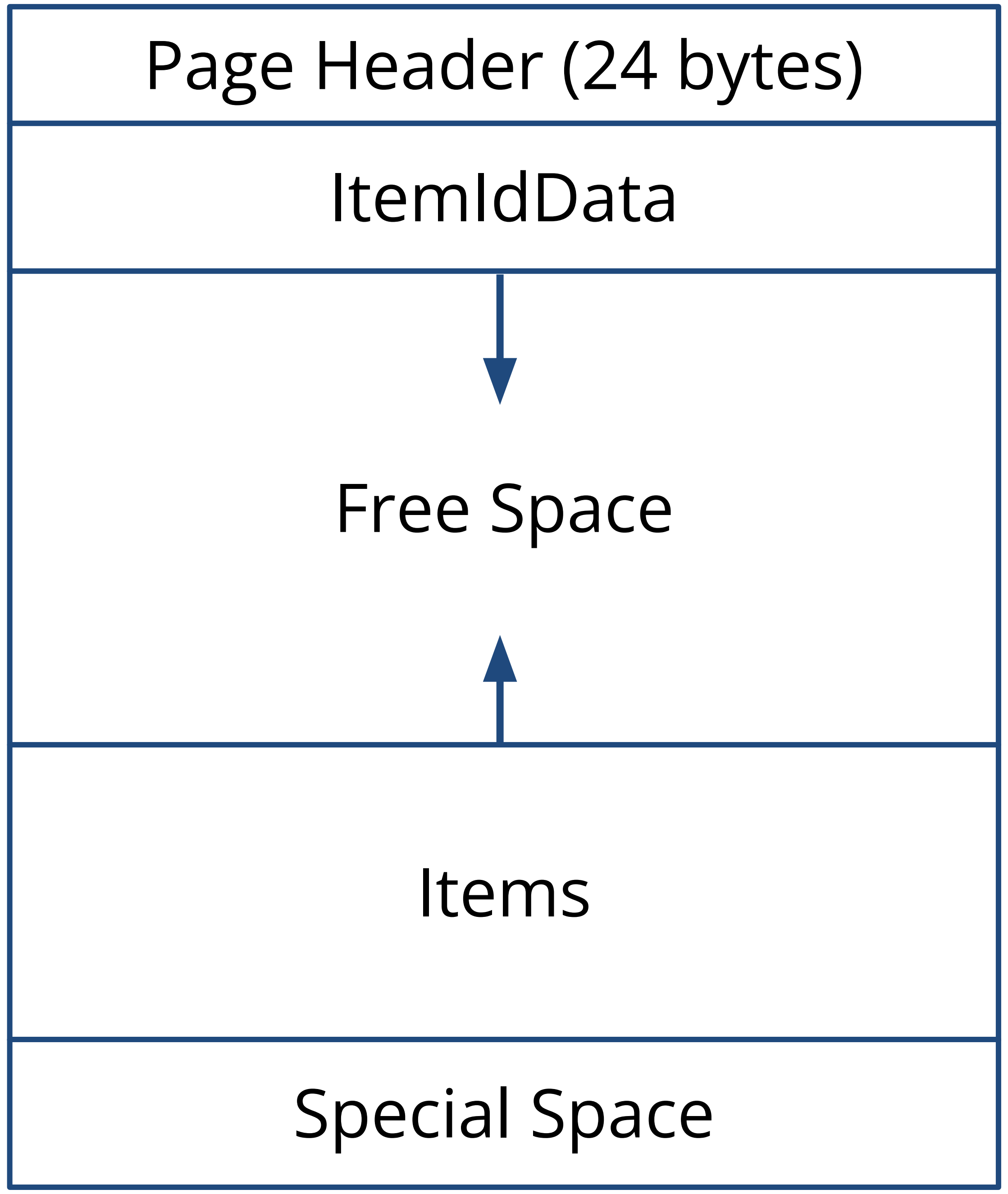
But wait.. there is more!

- Data driven - experiments are included
- Data driven, even if data is wrong (5%, 50%, 500%)
- Presentation notes include commands, and setup hints

Indexes are a tradeoff

- Indexes trade disk space and increased IO during changes for reduced IO when accessing existing records.
- PostgreSQL needs to decide to use an index
- Use explain on a query to check what PostgreSQL thinks is the best plan.

Data Page layout



Page Header (24 bytes)

ItemIdData

Free Space

Items

Special Space

```
SELECT relname, relkind,

reltuples / relpages AS
avg_tuples_per_page

FROM pg_class
WHERE
relpages>0 and reltuples>100;
```

Statistics

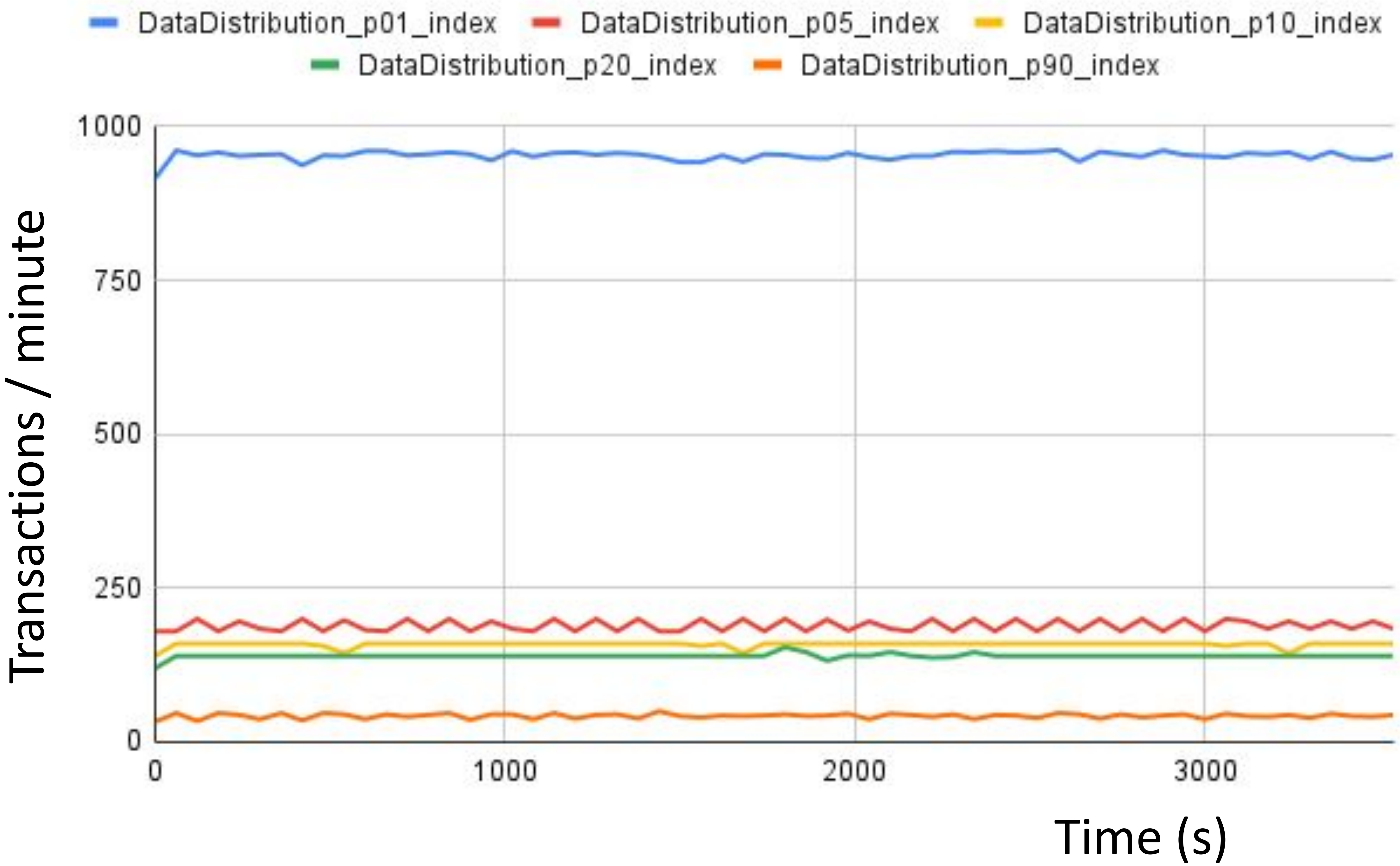
- pg_class -> reltuples /relpages
- pg_stat_all_tables -> last_autoanalyze, last_analyze, table usage
- pg_stats -> column frequent values and frequencies, avg width

```
SELECT  tablename, attname, avg_width,
        most_common_vals, most_common_freqs
FROM pg_stats
WHERE tablename='test' ;
```

Trade-offs

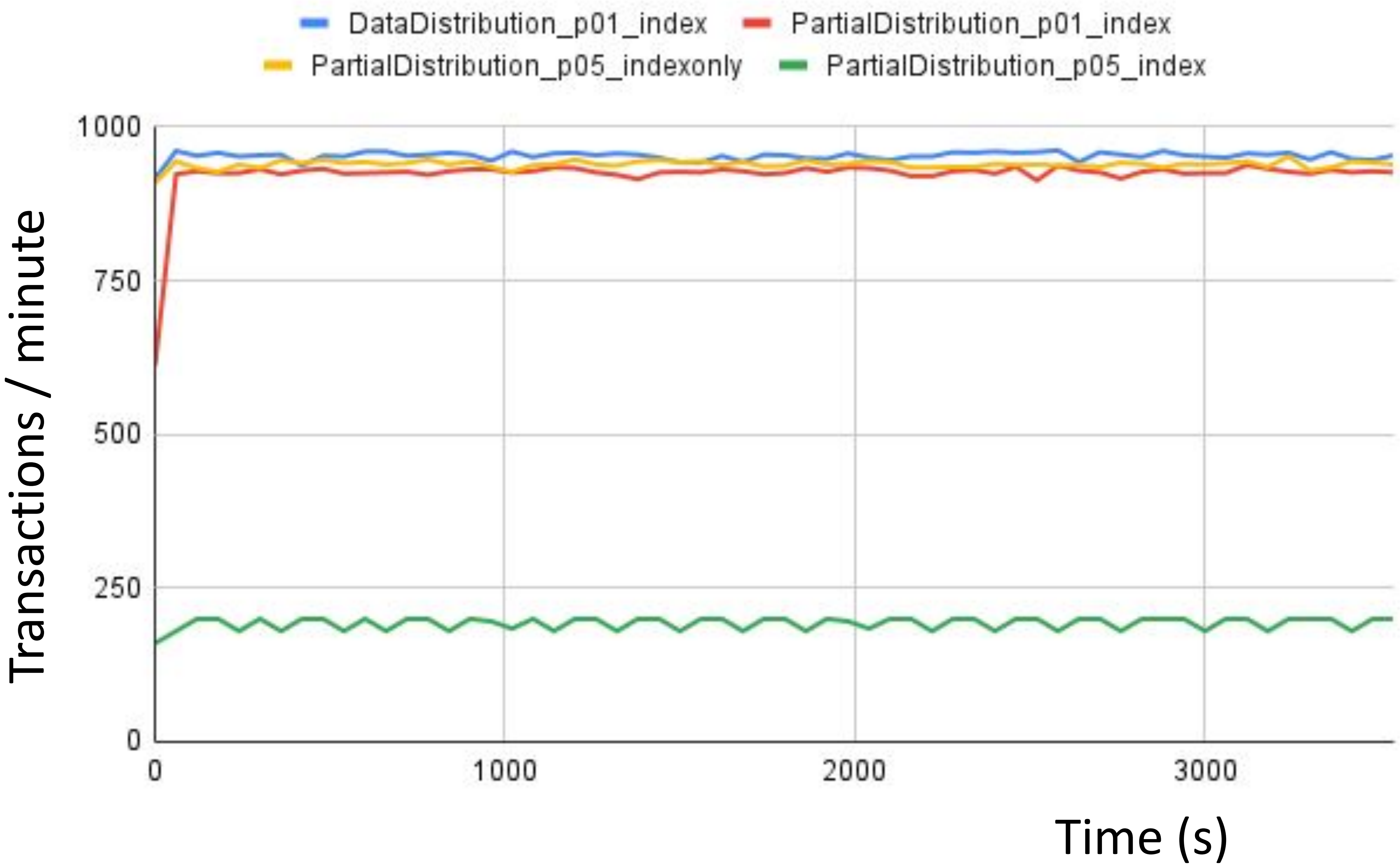
COMMAND PROMPT, INC.

Data distribution

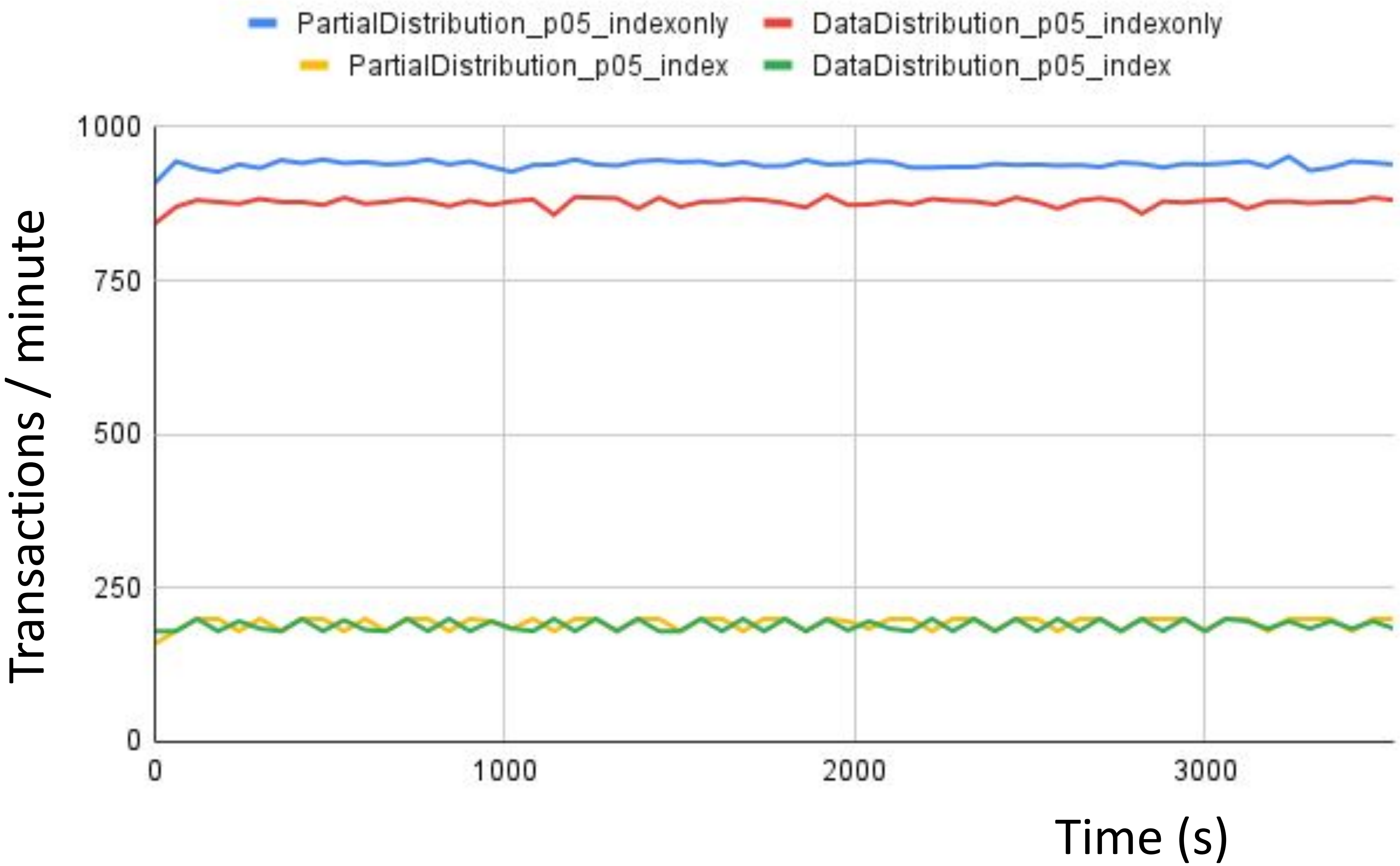


COMMAND PROMPT, INC.

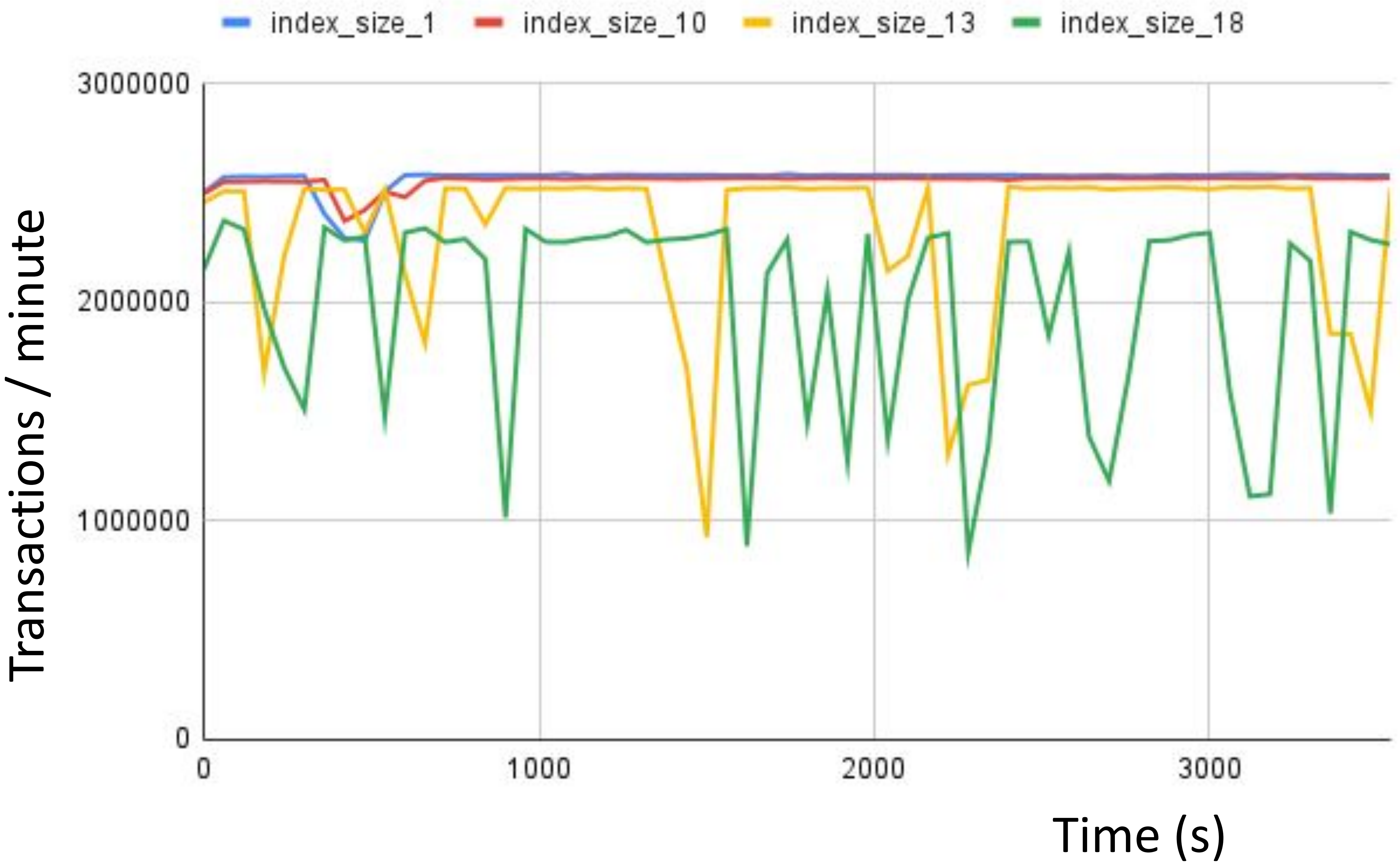
Data distribution - partial indexes



Index only access

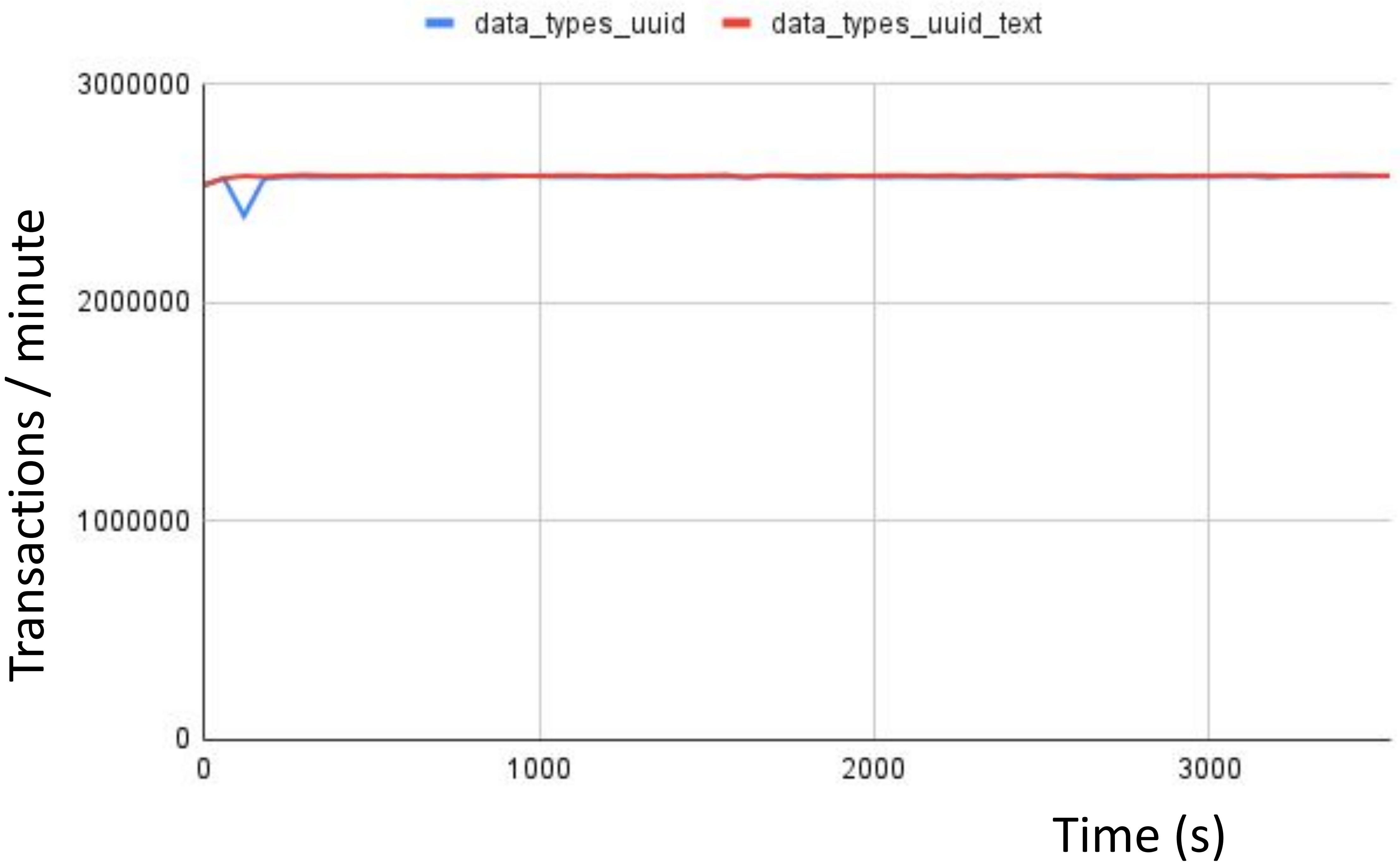


Index size

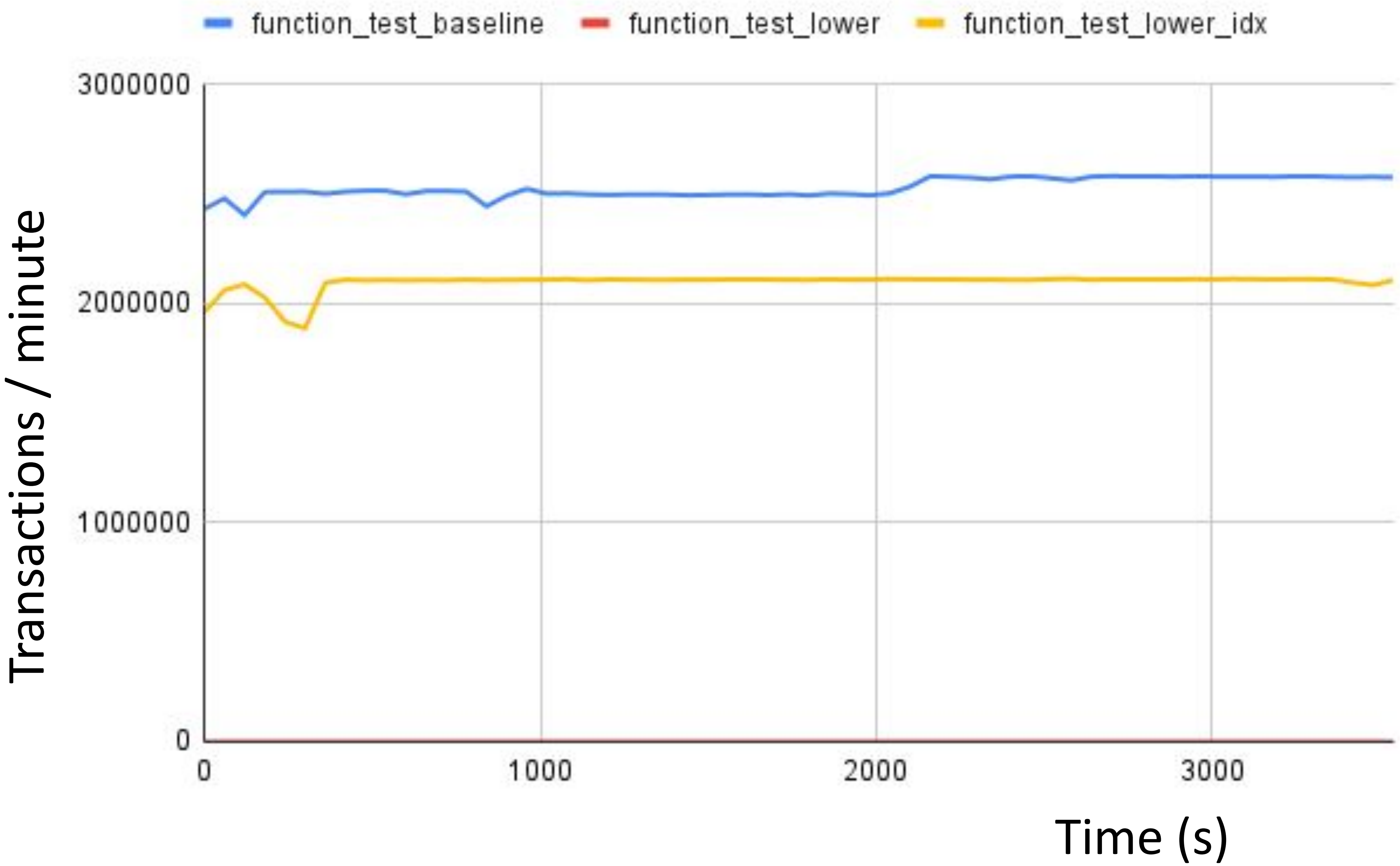


COMMAND PROMPT, INC.

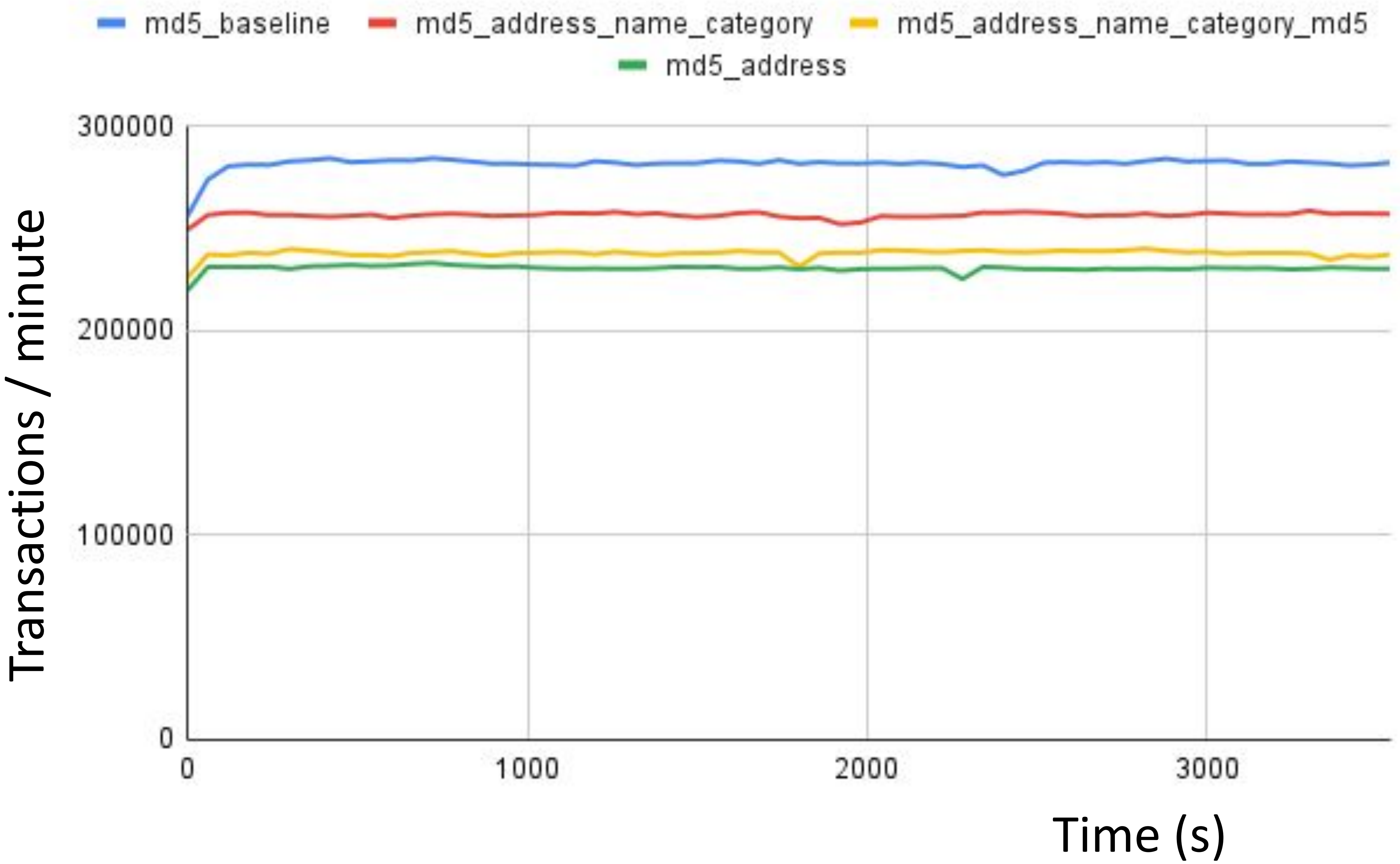
Data types


data_types_uuid
data_types_uuid_text
3000000
2000000
1000000
0
0
1000
2000
3000
Transactions / minute
Time (s)

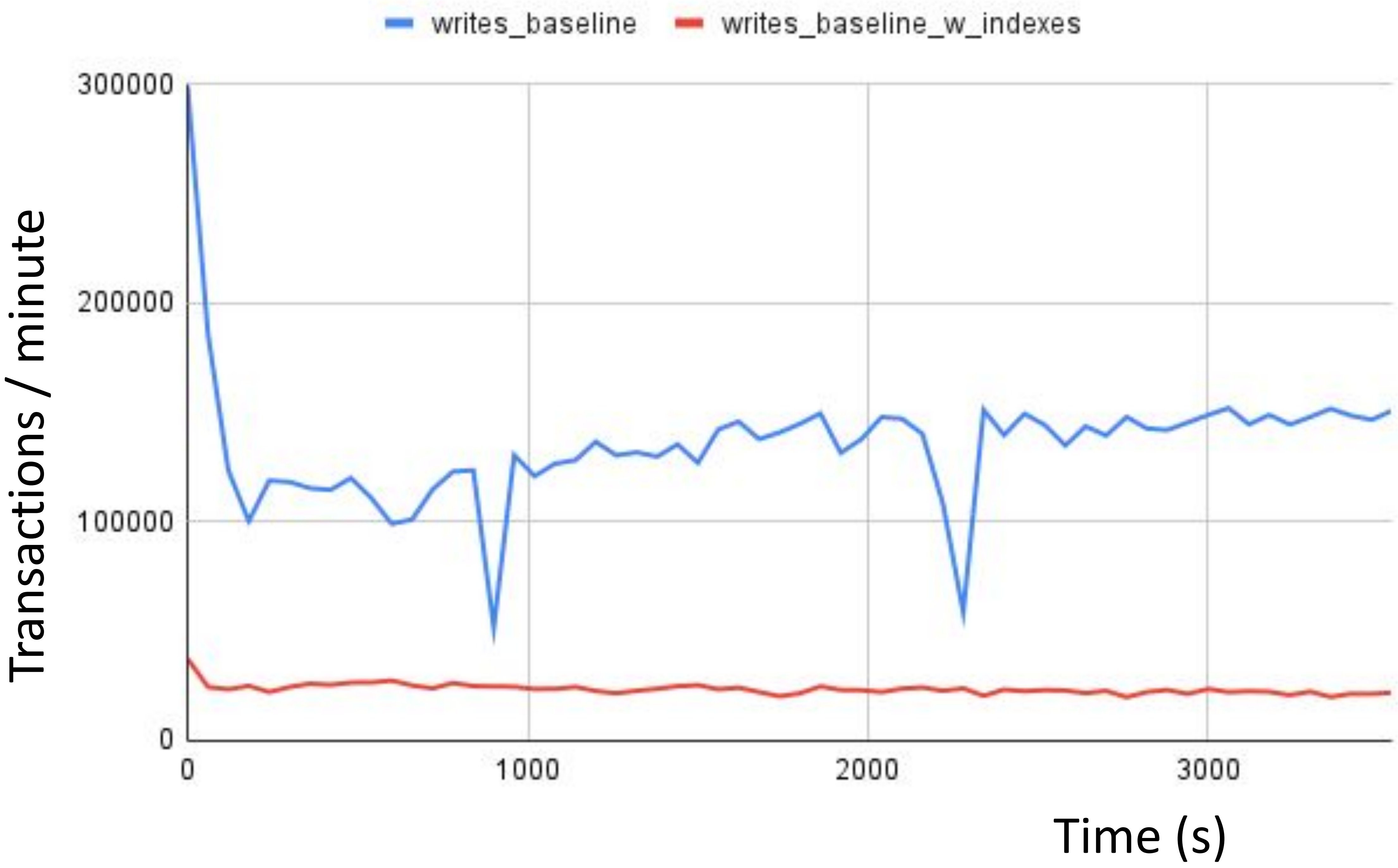
Functions are bad!



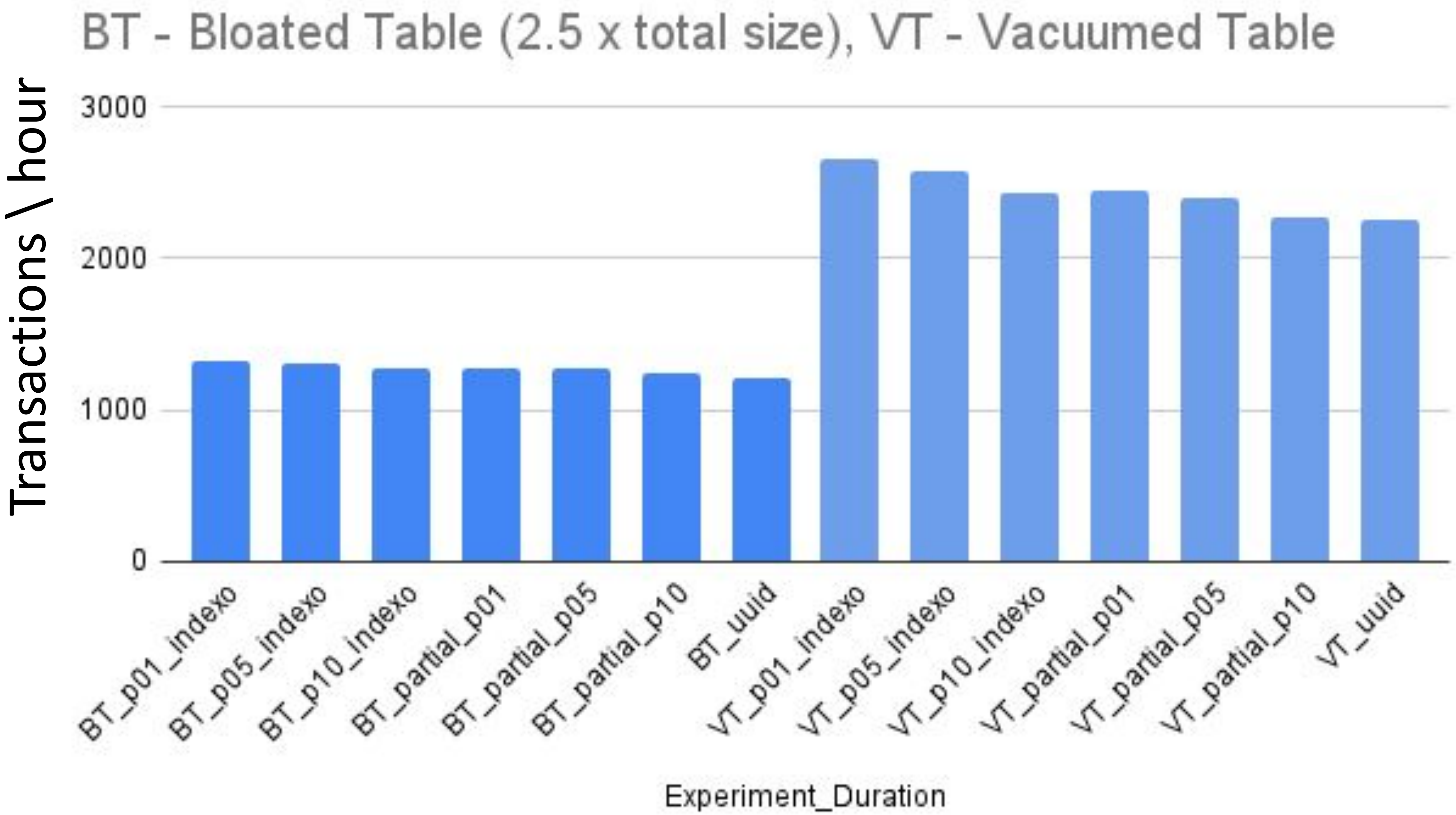
Functions are great!



Cost of writing



Maintenance / bloat



COMMAND PROMPT, INC.

Resources

COMMAND PROMPT, INC.

Test Table

```
CREATE TABLE test (
  id SERIAL PRIMARY KEY,          identifier uuid,
  text_identifier text,           category int,
  subcategory int,                p_01 int,
  p_05 int,                       p_10 int,
  p_15 int,                       p_20 int,
  p_25 int,                       p_50 int,
  p_75 int,                       p_90 int,
  name VARCHAR(50) NOT NULL,      email VARCHAR(100) NOT NULL,
  -- Other user-related columns
  bio TEXT,                       phone_number VARCHAR(20),
  address VARCHAR(200),           website_url VARCHAR(200),
  public_key TEXT );
```

Use the Index, Luke!

http://use-the-index-luke.com

COMMAND PROMPT, INC.

SUMMARY

COMMAND
PROMPT, INC.

Recap

- Everything is about trade-offs
- There are many factors including:
  - are statistics up to date?
  - shared_buffers / file cache
  - How many IO operations to access required data?

COMMAND PROMPT, INC.

Questions?

COMMAND
PROMPT, INC.

COMMAND PROMPT, INC.

+1 503 667 4564

www.commandprompt.com/contact-us

EXPERTS IN POSTGRES AND OPEN SOURCE INFRASTRUCTURE

Creating test table
Executing pgbench with custom code
Pgbench custom script strategy

Creating test table

This code will set up a testing table of any size. Unfortunately this code is not really all that efficient when it comes to creating millions of rows. That’s something I should improve at some point in the future.

In the meantime, it will work, eventually.

num_records=100000000 should result in an approximately 26 GB base table. At various times the indexes and bloat can result in a much larger database.

Unset

```
-- commandline example
-- psql -v num_records=1000 -f test_table.sql

-- Set a default value if num_records is not provided
-- Set a default value if num_records is not provided
-- Step 1:
\set num_records :num_records
-- If we defined num_records on commandline, it will just be set again
-- if it was not defined, it will be set to the string :num_records
-- Step 2:
SELECT CASE
  WHEN :'num_records'= ':num_records'
  THEN '100000000' --  800000000 -- switch to 100,000,000 records and re-do
  ELSE :'num_records'
END::numeric AS "num_records"  \gset

--TODO: add tables with common first names, last names, city names, etc.
-- use select first_name from rnd order by random();
```

```
create extension if not exists pgcrypto;
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE OR REPLACE FUNCTION generate_random_text(paragraphs INTEGER,
words_per_paragraph INTEGER)
RETURNS TEXT AS $$
DECLARE
  result TEXT := '';
  paragraph TEXT;
  word TEXT;
  i INTEGER;
  j INTEGER;
BEGIN
  FOR i IN 1..paragraphs LOOP
    paragraph := '';
    FOR j IN 1..words_per_paragraph LOOP
      word := '';
      FOR k IN 1..random() * 10 + 1 LOOP
        word := word || chr(65 + floor(random() * 26)::INTEGER);
      END LOOP;
      paragraph := paragraph || ' ' || word;
    END LOOP;
    result := result || paragraph || E'\n\n';
  END LOOP;
  RETURN result;
END;
$$ LANGUAGE plpgsql;



-- Base table

CREATE TABLE test (
  id SERIAL PRIMARY KEY,
  identifier uuid,
  text_identifier text,
  category int,
  subcategory int,
  p_01 int,
  p_05 int,
  p_10 int,
  p_15 int,
  p_20 int,
  p_25 int,
  p_50 int,
  p_75 int,
  p_90 int,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL,
  -- Other user-related columns
  bio TEXT,
  phone_number VARCHAR(20),
  address VARCHAR(200),
  website_url VARCHAR(200),
  public_key TEXT
);

INSERT INTO test (identifier, text_identifier, category, subcategory, p_01,
p_05, p_10, p_15,
p_20, p_25, p_50, p_75, p_90, name, email, address)
SELECT
  uuid_generate_v4() AS identifier,
  uuid_generate_v4()::text AS text_identifier,
  (random() * 10 + 1) AS category,
  (random() * 10 + 1) AS subcategory,
  CASE WHEN random() < 0.01 THEN 1 ELSE 0 END AS p_01,
  CASE WHEN random() < 0.05 THEN 1 ELSE 0 END AS p_05,
  CASE WHEN random() < 0.10 THEN 1 ELSE 0 END AS p_10,
  CASE WHEN random() < 0.15 THEN 1 ELSE 0 END AS p_15,
  CASE WHEN random() < 0.20 THEN 1 ELSE 0 END AS p_20,
  CASE WHEN random() < 0.25 THEN 1 ELSE 0 END AS p_25,
  CASE WHEN random() < 0.50 THEN 1 ELSE 0 END AS p_50,
  CASE WHEN random() < 0.75 THEN 1 ELSE 0 END AS p_75,
  CASE WHEN random() < 0.90 THEN 1 ELSE 0 END AS p_90,
  generate_random_text(1,3) AS name,
  generate_random_text(1,1) || '@' || generate_random_text(1,1) || '.com'  AS
email,
  generate_random_text(1,10) as address
FROM generate_series(1,:num_records) as i;
```

Executing pgbench with custom code

Figuring out which parameters can be with pgbench when executing custom code can involve a bit of trial and error. All my tests were performed with a command similar to this:

Unset

```
pgbench -d guide -f {test_file} -c 20 -j 4 -T 3600 -n -r -l
--log-prefix=output/{test_name} --aggregate-interval=60 >>
output/pgbench_{test_name}.out 2>/dev/null
```

Let’s go through the parameters:

-d guide	connect to the guide database
-f {test_file}	executes the desired test file (samples below)
-c 20	Run with 20 concurrent clients
-j 4	Each client should run 4 threads
-T 3600	Each test runs for 1 hour (3600 seconds)
-n	Do not run vacuum (I do those manually)
-r	Report average latency per command
-l	Write transaction logs to log file
–log-prefix=output/{test_name}	All outputs get put in the output directory with a specified test name
–aggregate-interval=60	Aggregate data every 60 seconds
>> output/pgbench_{test_name}.out	Save pgbench output to an output file
2>/dev/null	Do not save the runtime output.

Pgbench custom script strategy

Instead of giving you all the individual scripts, I’m just going to give a simplified guide to writing custom pgbench scripts.

Should you find results that do not make sense, please let me know. I’d love to find out if I made a mistake somewhere.

Accessing records via non integer field

At various times we need to access records by text or uuid column. In those cases I will generally create a random id, look up the field I need and then use that in my lookup or calculation.

When comparing multiple different scenarios, I will make sure that the corresponding variables are set in each, in order to make the id lookup consistent in all scenarios.

Unset

```
\set id random(1, 100000000)

select '''' || identifier::text || '''' as identifier from test where id=:id
\gset

select * from test where text_identifier=:identifier;
```

This technique can be abstracted to multiple values as well. For example, md5 lookup by multiple columns.

Unset

```
\set id random(1, 100000000)

select '''' || address || '''' as address , category as category, '''' || name
|| '''' as name from test where id=:id \gset

select * from test where md5( address || '!' || name || '!' ||
category::text)::uuid = md5(:address || '!' || :name || '!' ||
:category::text)::uuid;
```