Benchmark Test Results

A collection of the results of benchmark tests that I run on databases

Ben Brumm

www.databasestar.com

Benchmark Test Results

In this guide, you'll see the results of the benchmark tests I have run on databases.

This guide includes (for each test):

- a description of the test
- the results of the test, both numbers and charts
- an interpretation of the results.

Let's get right into it.

Note: this only has one benchmark result at the moment, but more will be added as more tests are run.

Benchmark 1: INT vs UUID Insert on PostgreSQL

This benchmark aims to answer the following question:

"Is it faster to insert an auto-incrementing integer or a UUID as a primary key in PostgreSQL?"

For this benchmark, I used PostgreSQL 16.8 on my MacBook Pro

I ran a series of Insert statements with four scenarios:

- Using a SERIAL primary key (INT)
- Using a GENERATED AS IDENTITY primary key (INT)
- Using a SEQUENCE object (INT)
- Using a UUID with the gen_random_uuid() function

Scripts

Here's the code to set up the tests, which uses the k6 load testing tool: https://k6.io

load_test_insert.js

```
import sql from 'k6/x/sql';
import driver from 'k6/x/sql/driver/postgres';

// Connection config - adjust to your Postgres setup
const db = sql.open(driver,
'postgres://username:password@localhost:5432/postgres_dev?sslmode=disabl
e');

export const options = {
  vus: 1,          // 10 virtual users running concurrently
  duration: '30s',  // run for 30 seconds
};

export function setup() {
  //Drop table if exists
  db.exec(`DROP TABLE IF EXISTS load_test_entries;`);

  // Create the table once before the test runs
  /*
  SERIAL:
  db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
```

```
      id    SERIAL PRIMARY KEY,
      name  TEXT NOT NULL,
      value INT  NOT NULL,
      created_at TIMESTAMPTZ DEFAULT NOW()
    );
  `);
  */

  //IDENTITY
  /*
    db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
      id    INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
      name  TEXT NOT NULL,
      value INT  NOT NULL,
      created_at TIMESTAMPTZ DEFAULT NOW()
    );
  `);
  */

  //SEQUENCE
  /*
    db.exec(`
      CREATE SEQUENCE IF NOT EXISTS load_test_entries_id_seq START 1
INCREMENT 1 NO MAXVALUE;
    `);

  db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
      id    INT PRIMARY KEY DEFAULT nextval('load_test_entries_id_seq'),
      name  TEXT NOT NULL,
      value INT  NOT NULL,
      created_at TIMESTAMPTZ DEFAULT NOW()
    );
  `);
  */

  //UUID type populated with gen_random_uuid() function
    //db.exec(`CREATE EXTENSION IF NOT EXISTS "pgcrypto";`);
    /*
    db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
        id    UUID PRIMARY KEY DEFAULT gen_random_uuid(),
        name  TEXT NOT NULL,
        value INT  NOT NULL,
        created_at TIMESTAMPTZ DEFAULT NOW()
    );
    `);
```

```
    */



}

export default function () {
  const name  = `user_${Math.floor(Math.random() * 10000)}`;
  const value = Math.floor(Math.random() * 1000);

  db.exec(`INSERT INTO load_test_entries (name, value) VALUES
('${name}', ${value});`);
}

export function teardown() {
  db.close();
}
```

load_test_iterations.js

```
import sql from 'k6/x/sql';
import driver from 'k6/x/sql/driver/postgres';

const db = sql.open(driver,
'postgres://username:password@localhost:5432/postgres_dev?sslmode=disabl
e');

export const options = {
  scenarios: {
    insert: {
      executor: 'shared-iterations',
      vus: 10,
      iterations: 1000000,  // exact row count
      maxDuration: '10m',   // safety cutoff in case something stalls
    },
  },
};

export function setup() {
  db.exec(`DROP TABLE IF EXISTS load_test_entries;`);

  //SERIAL
  /*
  db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
      id    SERIAL PRIMARY KEY,
      name  TEXT NOT NULL,
```

```
      value INT  NOT NULL,
      created_at TIMESTAMPTZ DEFAULT NOW()
    );
  `);
  */

  //IDENTITY
  /*
    db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
      id    INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
      name  TEXT NOT NULL,
      value INT  NOT NULL,
      created_at TIMESTAMPTZ DEFAULT NOW()
    );
  `);
  */

  //Sequence
  /*
  db.exec(`
      CREATE SEQUENCE IF NOT EXISTS load_test_entries_id_seq START 1
INCREMENT 1 NO MAXVALUE;
    `);

  db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
      id    INT PRIMARY KEY DEFAULT nextval('load_test_entries_id_seq'),
      name  TEXT NOT NULL,
      value INT  NOT NULL,
      created_at TIMESTAMPTZ DEFAULT NOW()
    );
  `);
  */

  //UUID type populated with gen_random_uuid() function


    db.exec(`
    CREATE TABLE IF NOT EXISTS load_test_entries (
        id    UUID PRIMARY KEY DEFAULT gen_random_uuid(),
        name  TEXT NOT NULL,
        value INT  NOT NULL,
        created_at TIMESTAMPTZ DEFAULT NOW()
    );
    `);

}
```

```
export default function () {
  const name  = `user_${Math.floor(Math.random() * 10000)}`;
  const value = Math.floor(Math.random() * 1000);

  db.exec(`INSERT INTO load_test_entries (name, value) VALUES
('${name}', ${value});`);
}

export function teardown() {
  db.close();
}
```

Results - Numbers

This table shows the results of each test scenario, run for different combinations of virtual users.

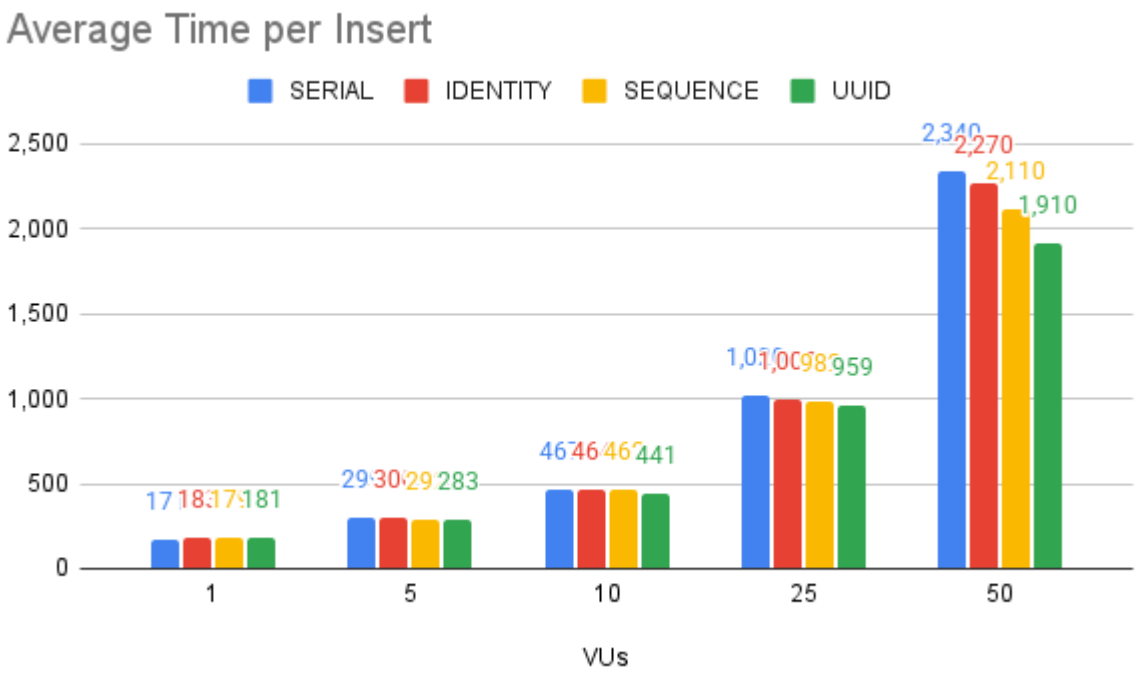
		Latency			Throughput	
Model	**VUs**	**Average µs**	**P(90) µs**	**p(95) µs**	**Iterations**	**Iterations Per Second**
SERIAL	1	170.80	206.41	234.58	170,909	5,657
SERIAL	5	299.28	371.83	446.50	491,758	16,357
SERIAL	10	467.36	572.45	695.43	633,331	21,017
SERIAL	25	1,020.00	1,420.00	2,190.00	727,698	24,202
SERIAL	50	2,340.00	5,280.00	7,890.00	637,010	21,115
IDENTITY	1	182.65	218.62	244.45	160,145	5,326
IDENTITY	5	299.77	364.79	417.91	502,088	16,702
IDENTITY	10	463.52	525.66	619.83	638,903	21,250
IDENTITY	25	1,000.00	1,320.00	2,100.00	743,543	24,728
IDENTITY	50	2,270.00	5,100.00	7,790.00	658,566	21,904
SEQUENCE	1	179.45	214.83	238.58	162,835	5,416
SEQUENCE	5	292.27	355.33	406.00	503,439	16,738
SEQUENCE	10	461.63	523.33	626.20	641,406	21,340
SEQUENCE	25	981.90	1,320.00	2,100.00	758,503	25,239
SEQUENCE	50	2,110.00	4,690.00	7,090.00	706,134	23,483
UUID gen_random_uuid	1	180.93	215.87	240.16	161,576	5,374
UUID	5	282.88	356.25	404.68	519,850	17,288

gen_random_uuid						
UUID gen_random_uuid	10	441.13	530.20	617.38	670,438	22,298
UUID gen_random_uuid	25	958.96	1,200.00	1,830.00	776,516	25,287
UUID gen_random_uuid	50	1,910.00	3,260.00	5,570.00	779,415	25,929

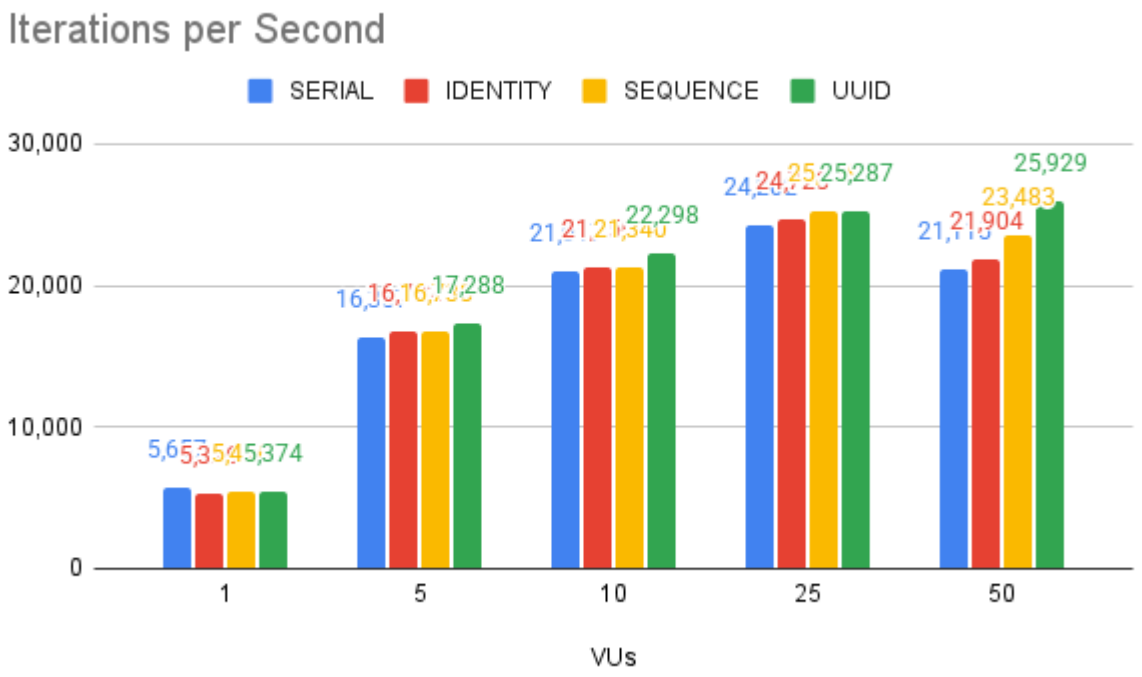
Results - Charts

Here are the charts of these results.

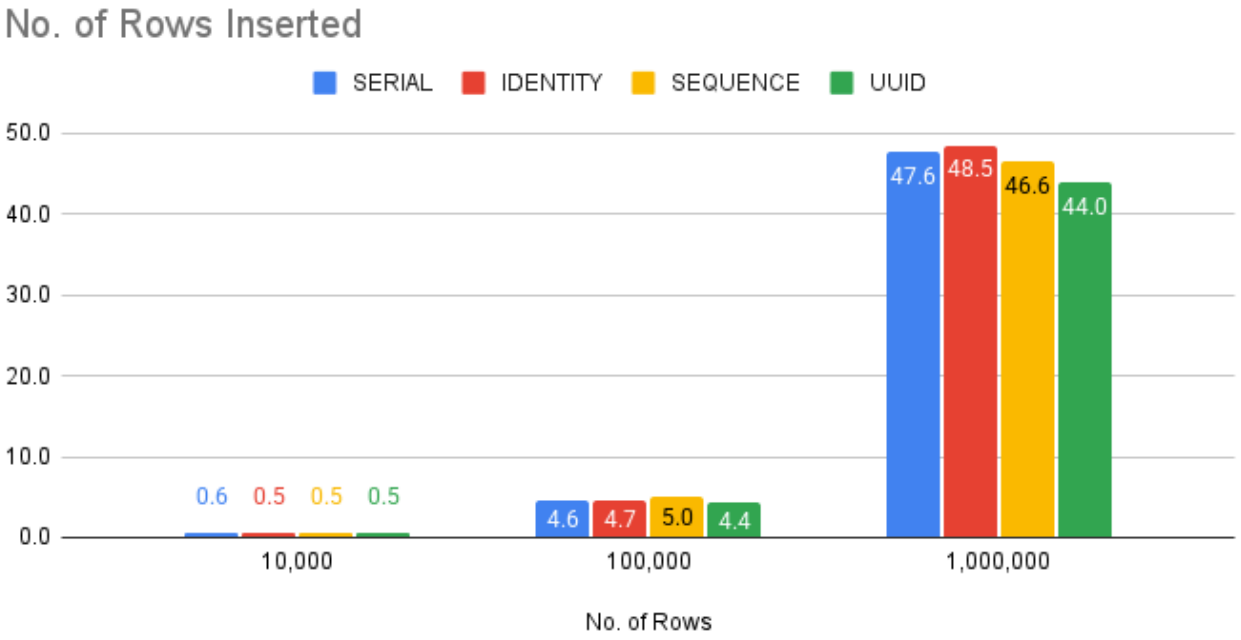
Average Time per Insert



Iterations Per Second



Time Taken for No. of Rows Inserted



Benchmark Learnings

From these results, we can see:

- UUID was the fastest to insert overall
- Out of the three INT tests, the SEQUENCE object performed better

Conclusion

I hope you found this useful. If you have any questions or requests for other benchmarks you want to see, let me know at ben@databasestar.com

Thanks,

Ben Brumm

www.DatabaseStar.com