

Fun With UUIDs

55554944-7320-4172-6520-436f6f6c2121

PGDay Lowlands 2025



Classes

Classes

This Week Next Week

Tuesday 05 September 2023

Intermediate

16:30 - 17:30
Kingston Dojo

Wednesday 06 September 2023

Advanced

17:00 - 18:00
Kingston Dojo

Tuesday 12 September 2023



Your Account

Your Membership

Pay As You Go

Come and train on an ad-hoc basis, pay per session, ideal for students starting out.

£25 per year - Membership Fee

£8 per class - Each class

Your next payment will be taken on

Pause Membership

Your Balance

Pay As You Go

Train as you need

Come and train on an ad-hoc basis, pay per session, ideal for students starting out.

2

Classes until
05 October 2023

Your Details

Hello John Smith

8th Kyu

Email: info@mokuso.cloud

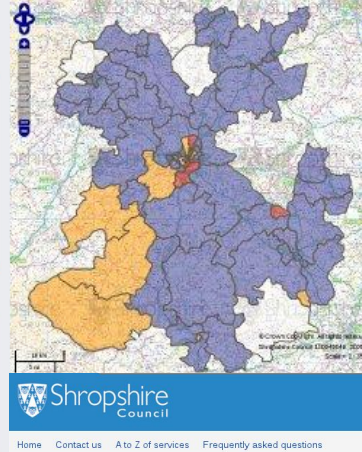
Mobile: 07848123456

Member since 05 September 2023

Change Details

Logout

Home Account Classes Account



Shropshire Council Family Information Directory Community Directory

Search

returned 98 results in 0.13 Seconds

☐ **63 (Bridgnorth) Squadron Air Training Corps**

Air Training Corps for young people between 13 and 20 years.

☐ **Albrighton Trust**

Albrighton Trust provides recreation and education for people with disabilities.

☐ **Apprenticeships**

Information about apprenticeships.

☐ **Archaeology Service**

The Archaeology Service aims to provide the archaeological heritage of the county

☐ **Ashley Music School**

Ashley Music School offers tuition in pian-electric and acoustic guitar, recorders an-

☐ **Bishop's Castle IT Centre**

Filter by category:

- ☒ education and learning
- ☐ leisure and culture
- ☐ community and living
- ☐ health and social care
- ☐ environment and planning
- ☐ jobs and careers
- ☐ business
- ☐ transport and streets
- ☐ advice and benefits



I'm Chris

IT jack of all trades,
studied Electronic
Engineering, mostly
a technical architect

Spend most of my
time building apps
a top PostgreSQL

Using PostgreSQL
for about ~20 years

UUID - What?



UUIDs

55554944-7320-4172-6520-436f6f6c2121

UUID Structure

55554944-7320-4172-6520-436f6f6c2121

&

00000000-0000-F000-F000-000000000000

=

00000000-0000-4000-6000-000000000000

Version

Variant

RFC 9562

UUID Data Type

```
CREATE TABLE club.subscription (  
    id            UUID,  
    member_id     UUID NOT NULL,  
    ...  
    CONSTRAINT subscription_pk PRIMARY KEY (id)  
);
```

Bad Press



UUID Performance

- Twice the size of int64
- Random allocation causes issues
 - Often more active index buffers
 - Extra Visibility Map lookups
 - Lost correlation

UUID Performance

Table "public.test_int"

Column	Type	Collation	Nullable	Default
id	bigint		not null	
value	text			

Indexes:

"test_int_pk" PRIMARY KEY, btree (id)

UUID Performance

Table "public.test_uuid"

Column	Type	Collation	Nullable	Default
id	uuid		not null	
value	text			

Indexes:

"test_uuid_pk" PRIMARY KEY, btree (id)

UUID Performance - Size

relname	pg_size_pretty
-----+-----	
test_int	5208 MB
test_int_rnd	5208 MB
test_uuid	5580 MB
test_int_pk	214 MB
test_int_rnd_pk	281 MB
test_uuid_pk	384 MB

UUID Performance - Size - Reindex

relname	pg_size_pretty
-----+-----	
test_int	5208 MB
test_int_rnd	5208 MB
test_uuid	5580 MB
test_int_pk	214 MB
test_int_rnd_pk	214 MB
test_uuid_pk	301 MB

UUID Performance - Buffers

EXPLAIN (BUFFERS, ANALYSE)

```
SELECT count(id) FROM test_int;
```

Index Only Scan using test_int_pk

Buffers: shared hit=27346

UUID Performance - Buffers

EXPLAIN (BUFFERS, ANALYSE)

```
SELECT count(id) FROM test_uuid;
```

Index Only Scan using test_uuid_pk

Buffers: shared hit=9545782

UUID Performance

EXPLAIN (BUFFERS, ANALYSE)

```
SELECT count(id) FROM test_int_rnd;
```

Index Only Scan using test_int_rnd_pk

Buffers: shared hit=9551903

So Why Use UUIDs?



Why UUIDs - Security

- Exposing sequential ids is not ideal
 - Against lots of security guidelines
- 2^{128} is a lot of combinations
 - Extremely hard to brute-force

Why UUIDs - Application

- Sequences are DB generated
 - `INSERT ... RETURNING id`
- UUIDs can be generated in the app
 - Across all nodes
 - Before insert

Why UUIDs - Migration / Sharding

- UUIDs are universally unique
 - Probably
- Simplifies
 - Sharding
 - Migrations

Why UUIDs - Bitwise Encoding

- Encode information inside the UUID
 - Tenant Id
 - Cluster Id
 - Partitioning
 - Entity type

Why UUIDs - Multi-Tenant

1b89ed15-0913-4ffa-8000-000000000000

|

00000000-0000-4000-8edb-347cff36bad2

=

1b89ed15-0913-4ffa-8edb-347cff36bad2

Event Smarter

72a500f1-211f-4001-81c3-ed36b0748e0f

Chunk

- Tenant
- Quarter

Time

Tag &
Seq

Random Id

Best Of Both Worlds?



UUIDv7

72a500f1-211f-7001-81c3-ed36b0748e0f

Time

Random

FLUUID - Flexible UUIDs - Time Based

000139a9-7064-8064-8000-9fba0fd0f203

Time

Tenant

Tag &
Seq

Random

00b1fb1f-d7f1-87f1-a1ce-853d2ec1e0a8

Time

Random

Tag

Random

FLUUID - Flexible UUIDs - Block Based

000139a9-7064-8064-8ce8-9fba0fd0f203

Block Tag Tenant

Random

FLUID Extended Performance

EXPLAIN (BUFFERS, ANALYSE)

```
SELECT count(id) FROM test_fluid;
```

Index Only Scan using test_fluid_pk

Buffers: shared hit=38322

FLUID Multi-Tenant Performance

EXPLAIN (BUFFERS, ANALYSE)

```
SELECT count(id) FROM test_fluid_mt;
```

Index Only Scan using test_fluid_mt_pk

Buffers: shared hit=714222

FLUID Multi-Tenant Performance

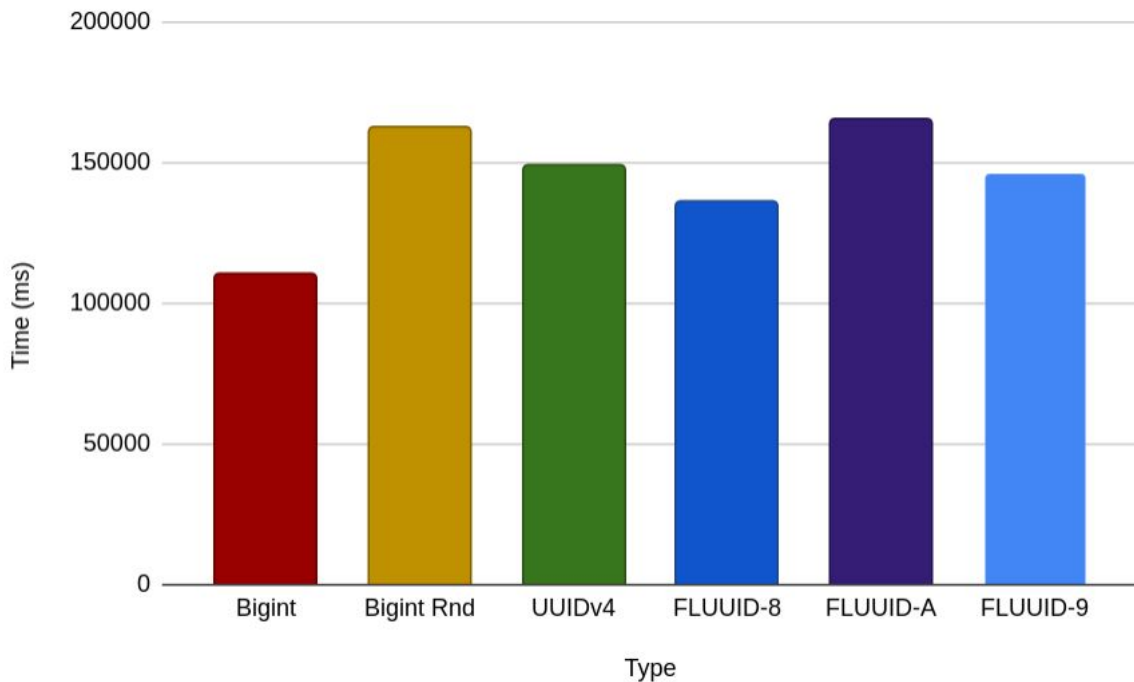
EXPLAIN (BUFFERS, ANALYSE)

```
SELECT count(id) FROM test_fluid_mt9;
```

Index Only Scan using test_fluid_mt9_pk

Buffers: shared hit=47545

Insert Performance - From Application



- Total time to insert 10M records
- From Java application
- All ids generated in application
- Records are id plus 500 byte data
- Starting from empty table with index

The Fun Part!



Generating Inside PostgreSQL

```
CREATE OR REPLACE FUNCTION
```

```
    fluuid_gen_variant8(p_tenant BIGINT, p_tag INTEGER, p_sequence INTEGER)
```

```
RETURNS UUID VOLATILE LANGUAGE SQL AS $$
```

```
SELECT (
```

```
    lpad(to_hex(((extract(epoch FROM clock_timestamp())-1735689600)/300)::BIGINT), 6, '0') ||
```

```
    lpad(to_hex((p_tenant >> 28) & 0xFF), 2, '0') ||
```

```
    '-' || lpad(to_hex((p_tenant >> 12) & 0xFFFF), 4, '0') ||
```

```
    '-8' || lpad(to_hex(p_tenant & 0xFFF), 3, '0') ||
```

```
    '-8' || lpad(to_hex(((p_tag & 0x3F) << 6 | (p_sequence & 0x3F)) & 0xFFF), 3, '0') ||
```

```
    '-' || right(gen_random_uuid()::TEXT, 12)
```

```
)::UUID;
```

```
$$;
```

Check That Fun



Extract UUID Version - PGSQL ;)

```
CREATE OR REPLACE FUNCTION fluuid_extract_version(p_id UUID)
RETURNS INTEGER IMMUTABLE LANGUAGE PLPGSQL AS $$
    DECLARE
        v_chunk      INTEGER;
    BEGIN
        EXECUTE 'SELECT x' || quote_literal(
            right(left(p_id::TEXT, 15), 1)
        ) || '::~INTEGER;' INTO v_chunk;
        RETURN v_chunk;
    END;
$$;
```

Banning gen_random_uuid() - ;)

```
ALTER TABLE test_fluid ADD CONSTRAINT uuid_version_chk  
CHECK (fluid_extract_version(id) = 8);
```

```
baddev=> INSERT INTO test_fluid (id, value)  
VALUES (gen_random_uuid(), 'Test');
```

```
ERROR:  new row for relation "test_fluid" violates  
check constraint "uuid_version_chk"
```

Extract UUID Variant - PGSQL ;)

```
CREATE OR REPLACE FUNCTION fluuid_extract_variant(p_id UUID)
RETURNS INTEGER IMMUTABLE LANGUAGE PLPGSQL AS $$
    DECLARE
        v_chunk      INTEGER;
    BEGIN
        EXECUTE 'SELECT x' || quote_literal(
            right(left(p_id::TEXT, 20), 1)
        ) || '::~INTEGER;' INTO v_chunk;
        RETURN v_chunk;
    END;
$$;
```

Ensuring We Have A Variant

```
ALTER TABLE test_fluuid_mt ADD CONSTRAINT uuid_version_mt_chk  
    CHECK (fluuid_extract_version(id) = 8 AND  
           fluuid_extract_variant(id) = 8);
```

```
baddev=> INSERT INTO test_fluuid_mt (id, value)  
          VALUES ('008dfbfb-e957-857b-a1ce-350c1966e3d9', 'Test');
```

```
ERROR:  new row for relation "test_fluuid_mt"  
        violates check constraint "uuid_version_mt_chk"
```

Time For Fun



Decoding Inside PostgreSQL

```
CREATE OR REPLACE FUNCTION fluuid_extract_time(p_id UUID)
RETURNS INTEGER IMMUTABLE LANGUAGE PLPGSQL AS $$
    DECLARE v_chunk      INTEGER;
    BEGIN
        EXECUTE 'SELECT x' || quote_literal(
            left(p_id::TEXT, 6)
        ) || '>::INTEGER;' INTO v_chunk;
        RETURN v_chunk;
    END;
$$;
```

Query By Time

```
SELECT count(id)
  FROM test_fluuid_part
 WHERE
    id >= '00000000-0000-8000-8000-000000000000'::UUID
 AND
    id < '00060000-0000-8000-8000-000000000000'::UUID;
```

Partition By Time

```
CREATE TABLE test_fluid_part  
  (id UUID, value TEXT)
```

```
PARTITION BY RANGE  
  (id);
```

Partition By Time

```
CREATE TABLE test_fluuid_part_251
PARTITION OF test_fluuid_part
    (CONSTRAINT test_fluuid_part_251_pk PRIMARY KEY (id))
FOR VALUES
FROM (fluuid_for_time('2025-01-01T00:00:00+0'::TIMESTAMPTZ))
TO (fluuid_for_time('2025-02-01T00:00:00+0'::TIMESTAMPTZ));
```

Funny Tenants



Sharding - Tenant Routing

```
// decode the TenantId from the UUID
TenantId tenantId = FluuidDecoder.tenant(id);
// select the database for the tenantId
Shard shard = ShardResolver.resolveShard(tenantId);
// connect to the shard
try (Connection con = shard.connect())
{ ... }
```

Decoding Inside PostgreSQL

```
CREATE OR REPLACE FUNCTION fluuid_extract_tenant(p_id UUID)
RETURNS BIGINT IMMUTABLE LANGUAGE PLPGSQL AS $$
    DECLARE    v_chunk        BIGINT;
    BEGIN
        EXECUTE 'SELECT x' || quote_literal(
            left(right(p_id::TEXT, 31), 2) || left(right(p_id::TEXT, 27), 4) ||
            left(right(p_id::TEXT, 21), 3)
        ) || '::BIGINT;' INTO v_chunk;
        RETURN v_chunk;
    END;
$$;
```

Query By Tenant

```
SELECT count(id)
FROM test_fluuid_mt
WHERE
    fluuid_extract_tenant(id) = 0xCECECECECE;
```

Query By Tenant

```
CREATE INDEX test_fluid_tenant_idx  
ON test_fluid_mt  
  (fluid_extract_tenant(id));
```

Partition By Tenant

```
CREATE TABLE test_fluuid_mt_part  
  (id UUID, value TEXT)  
PARTITION BY  
  LIST (fluuid_extract_tenant(id));
```

Partition By Tenant

```
CREATE TABLE test_fluid_mt_part_d  
    PARTITION OF test_fluid_mt_part  
    (CONSTRAINT test_fluid_mt_part_d_pk  
        PRIMARY KEY (id))  
  
DEFAULT;
```

Partition By Tenant

```
CREATE TABLE test_fluuid_mt_part_0  
    PARTITION OF test_fluuid_mt_part  
    (CONSTRAINT test_fluuid_mt_part_0_pk  
        PRIMARY KEY (id))  
    FOR VALUES IN (0x1CECECECE);
```

Partition By Tenant

```
CREATE TABLE test_fluuid_mt_part_0  
    PARTITION OF test_fluuid_mt_part  
    (CONSTRAINT test_fluuid_mt_part_0_pk  
        PRIMARY KEY (id))  
FOR VALUES IN (0x1CECECECE);
```

Down The Rabbit Hole - Extensions



Extension - C

```
PG_FUNCTION_INFO_V1(fluid_extract_time);
Datum fluid_extract_time(PG_FUNCTION_ARGS)
{
    int64      time;
    pg_uuid_t  *uuid = PG_GETARG_UUID_P(0);
    time = (((int64) uuid->data[0]) << 16) |
           (((int64) uuid->data[1]) << 8 ) |
           (((int64) uuid->data[2]));
    PG_RETURN_INT64(time);
}
```

Extension - C

```
PG_FUNCTION_INFO_V1(fluid_extract_tenant);
Datum fluid_extract_tenant(PG_FUNCTION_ARGS)
{
    int64      tenant;
    pg_uuid_t  *uuid = PG_GETARG_UUID_P(0);
    tenant = (((int64) uuid->data[3]) << 28) |
             (((int64) uuid->data[4]) << 20) |
             (((int64) uuid->data[5]) << 12) |
             (((int64) (uuid->data[6] & 0x0F)) << 8) |
             (((int64) uuid->data[7]));
    PG_RETURN_INT64(tenant);
}
```

Extension - Registering Your Functions

```
CREATE FUNCTION fluuid_extract_time(val UUID) RETURNS BIGINT  
AS 'MODULE_PATHNAME', 'fluuid_extract_time'  
IMMUTABLE LANGUAGE C STRICT PARALLEL SAFE;
```

```
CREATE FUNCTION fluuid_extract_tenant(val UUID) RETURNS BIGINT  
AS 'MODULE_PATHNAME', 'fluuid_extract_tenant'  
IMMUTABLE LANGUAGE C STRICT PARALLEL SAFE;
```

Extension - Speed Up

```
cellis=> INSERT INTO ext_test (SELECT fluuid_gen_variant8(0x1cecece, 0, 0)
      FROM generate_series(0, 1000000, 1));
```

```
INSERT 0 1000001
```

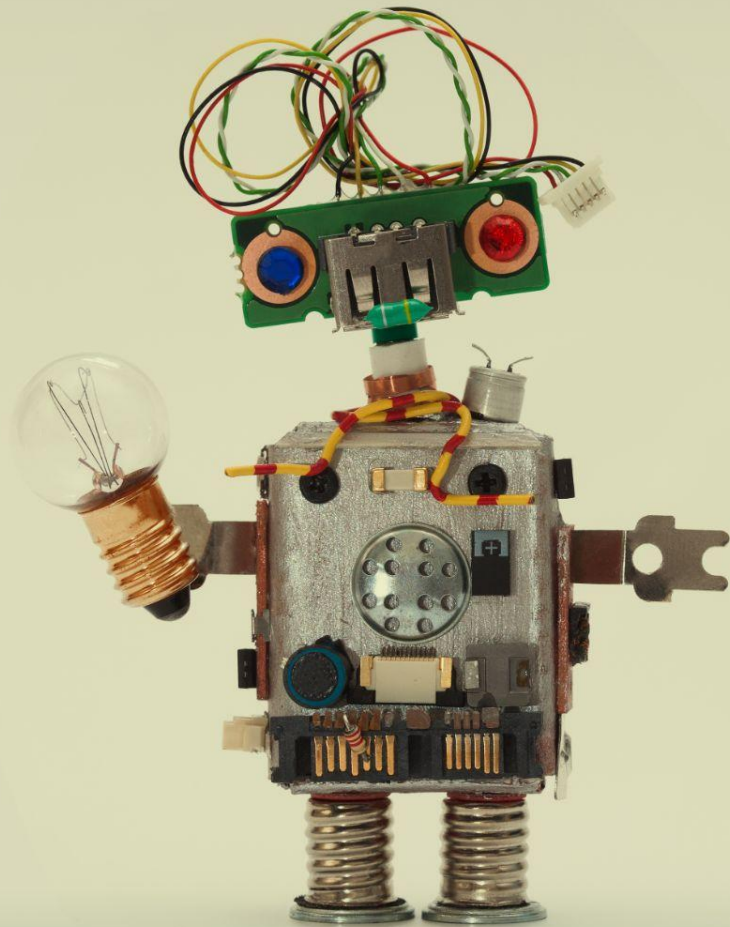
Time: 2729.613 ms (00:02.730)

```
cellis=> INSERT INTO ext_test (SELECT ext.fluuid_generate_v8(0x1cecece, 0, 0)
      FROM generate_series(0, 1000000, 1));
```

```
INSERT 0 1000001
```

Time: 2036.965 ms (00:02.037)

OR 25% faster



fluid.dev
(coming soon)
&
**Thanks For
Listening**