Auto ID Generation in PostgreSQL and YugabyteDB

Brett Hoyer

Friday, Jan/20/2023

YFTT YugabyteDB Friday Tech Talks

yugabyteDB

Approaches in PostgreSQL

1. Auto-incrementing
2. Sequence-caching
3. Client-side ID management
4. UUID-generation

Approach 1: Auto-incrementing

```
CREATE TABLE users(
     id SERIAL PRIMARY KEY,
     name VARCHAR(50)
);


// Equivalent

CREATE SEQUENCE users_id_seq;

CREATE TABLE users (
     id integer PRIMARY KEY DEFAULT nextval('users_id_seq'),
     name VARCHAR(50)
);

ALTER SEQUENCE users_id_seq OWNED BY users.id;
```

Approach 1: Auto-incrementing

Pros

- Developer friendly
- Sequential
- Atomic
- Storage size
  - SMALLSERIAL
    - 2 bytes (1 to 32767)
  - SERIAL
    - 4 bytes (1 to 2147483647)
  - BIGSERIAL
    - 8 bytes (1 to 9223372036854775807)
- Serial type or identity column

Cons

- Extra round trip to obtain next value from sequence
- Potential security vulnerabilities due to predictable nature of identifiers
- Can cause difficulties when importing data
- Sequential

Approach 2: Sequence-caching

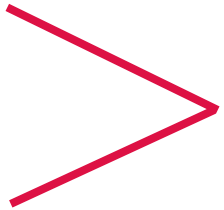
We can improve latencies by caching sequence values.

```
CREATE SEQUENCE IF NOT EXISTS users_id_seq
     START 1
     CACHE 100;
```

```
// DB Session 1
> SELECT NEXTVAL('users_id_seq');
1
> SELECT NEXTVAL('users_id_seq');
2
…
> SELECT NEXTVAL('users_id_seq');
100
```

```
// DB Session 2
> SELECT NEXTVAL('users_id_seq');
101
> SELECT NEXTVAL('users_id_seq');
102
…
> SELECT NEXTVAL('users_id_seq');
200
> SELECT NEXTVAL('users_id_seq');
201
```

```
> SELECT NEXTVAL('users_id_seq');
301
```



Approach 2: Sequence-caching

Pros

- Values cached in memory to improve latencies

Cons

- Gaps in sequence during dropped sessions, transaction rollbacks and database shutdowns

Approach 3: Client-side ID management

We can handle IDs client-side using a PostgreSQL sequence.

```
CREATE SEQUENCE IF NOT EXISTS users_id_seq
     START 1
     INCREMENT BY 50;
```

Approach 3: Client-side ID management

```
// Caller initialized with next value from sequence
let idVal = await pg.query("nextval('users_id_seq')");
let limit = idVal + 50;
. . .
// Obtain next value if we've exhausted all IDs
if (idVal >= limit) {
  idVal = await pg.query("nextval('users_id_seq')");
  limit = idVal + 50;
}
const text = 'INSERT INTO users(id, name) VALUES($1, $2) RETURNING *';
const values = [idVal, 'brett'];
await pg.query(text, values);
idVal += 1;
```

Approach 3: Client-side ID management

Pros

- Values can be prefetched from sequence to improve performance

Cons

- Gaps in sequence during application failures
- Application overhead and complexity

Approach 4: UUID-generation

- Algorithmically-generated Universally Unique Identifier
  - Example: **`7da1082a-65c0-46ce-8241-f8390022c099`**
- PostgreSQL has built-in functions for generating UUIDs
  - gen_random_uuid function returns V4 UUID
  - uuid_ossp module provides additional functionality for generating different types of UUIDs if required
- Can be generated client-side
- To use UUIDs in Yugabyte, install the *pgcrypto* extension

```
CREATE EXTENSION IF NOT EXISTS "pgcrypto";

CREATE TABLE users(
     id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
     name VARCHAR(50)
);
```

Approach 4: UUID-generation

Pros

- Nonsequential
- Can be generated on DB side or client-side
- Decentralized nature can improve latencies
- Easier to handle data import

Cons

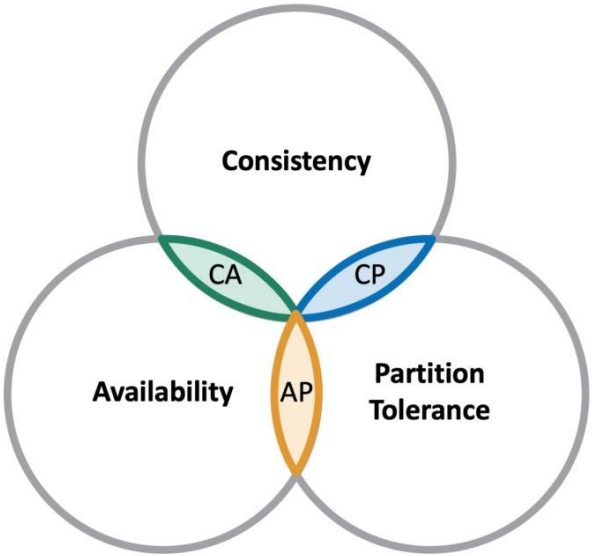
- Storage size
  - Typically 16 bytes
- Algorithms take time and resources to compute
- Uniqueness not 100% guaranteed, although an infinitesimally small likelihood of repeating

Demo Time

Multi-Region E-Commerce Application

Designing distributed SQL databases

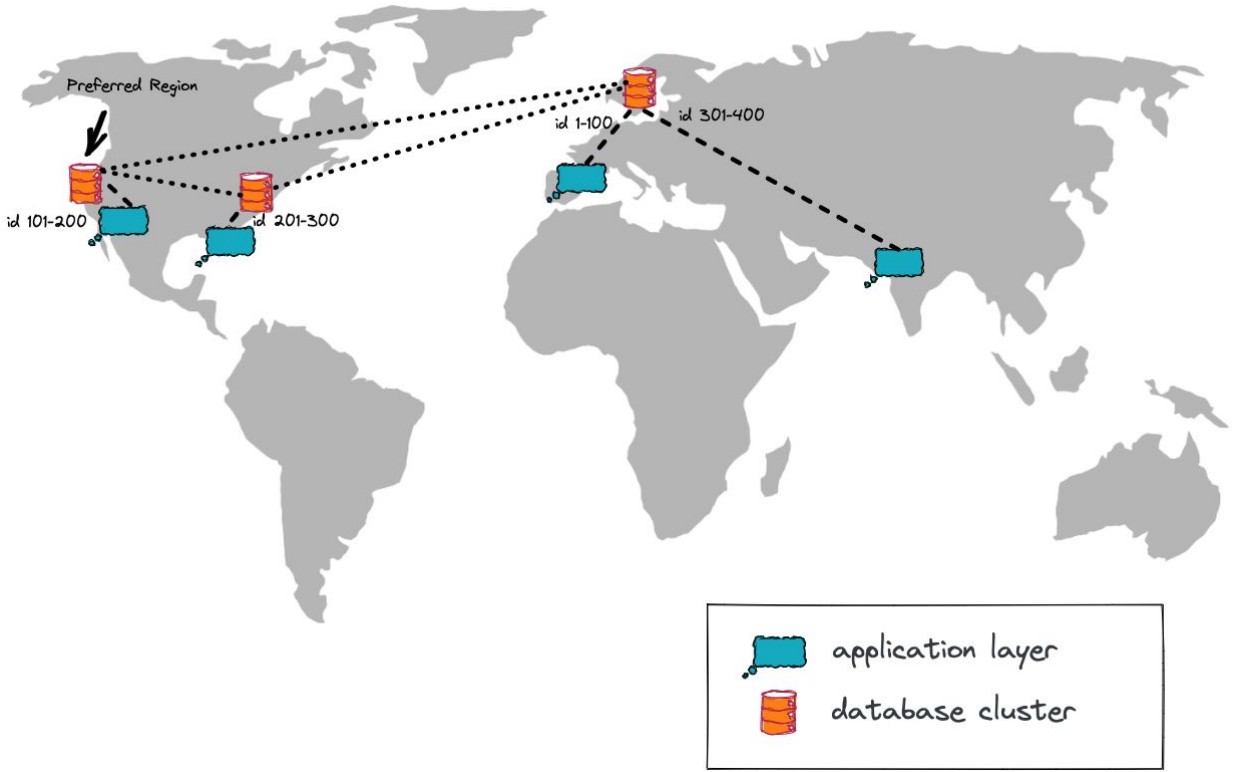
Cap Theorem



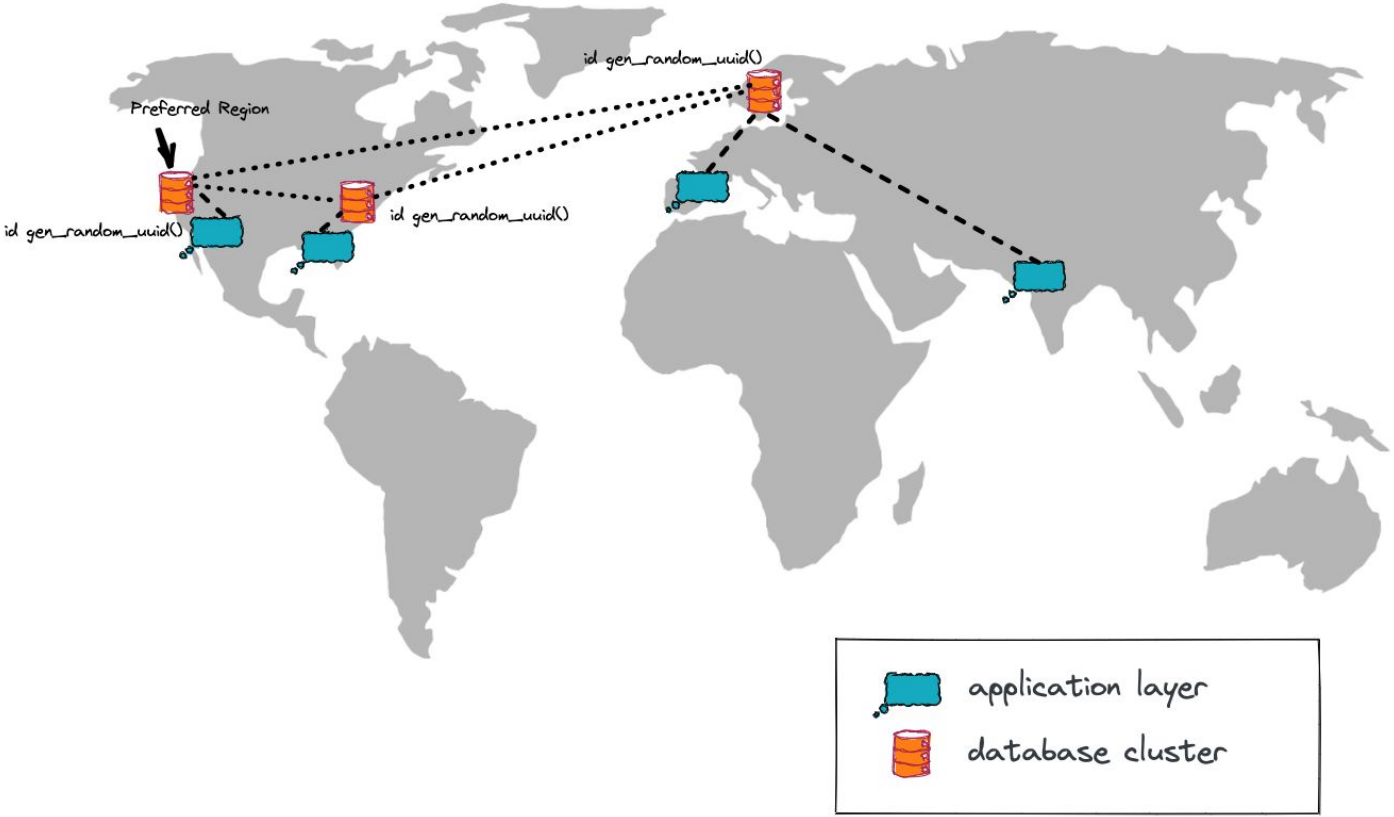
Raft Consensus Algorithm

A distributed SQL database uses the Raft consensus algorithm for both leader election and data replication. Instead of having a single Raft group for the entire dataset in the cluster, a distributed SQL database applies Raft replication at an individual shard level where each shard has a Raft group of its own.

Approach 2: Sequence-caching



Approach 4: UUID-generation



Thank You

Join us on Slack: yugabyte.com/slack (#yftt channel)

Star us on Github: github.com/yugabyte/yugabyte-db

YFTT YugabyteDB Friday Tech Talks

yugabyteDB