Happy hacking in Tantan using Golang & PostgreSQL

PostgreSQL in Tantan

Henry Ren
探探科技

How Tantan works

Relationships in Tantan

swipes

600 billion swipes in total

700 million swipes / day

900 million swipes / day (June 17)

passbys

200 billion passbys in total

Why PostgreSQL

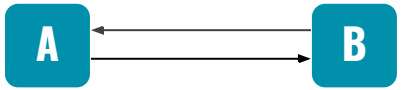
- "The world's most advanced open source database"
  - "It has more than 15 years of active development"
  - "It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (reliability, data integrity, and correctness)"
- PostGIS for Location Based Services (PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL)
  - Nearby users
  - Passby users
  - Distance with a user

Agenda

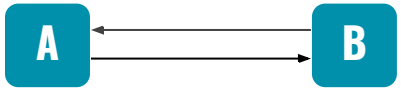
- Scaling Swipes
- PostgreSQL in Tantan

Scaling Swipes

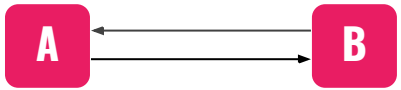
#1 How Swipe works



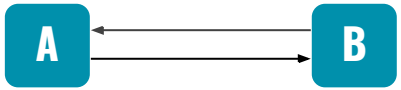
A liked B



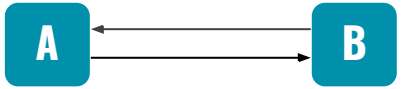
B liked A



A liked B
B liked A



A disliked B



B disliked A

#2 Product Requirements

- One user should only swipe another user once
- Mutual likes will create a Match
- Calculating nearby users
  - Liked users should be ranked higher
  - Disliked users must be filtered out

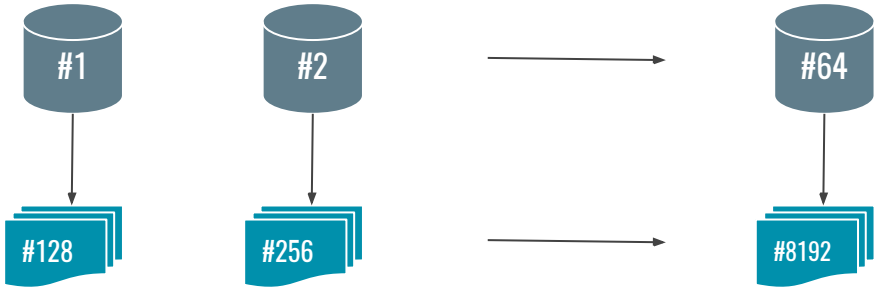
```
CREATE TYPE status AS ENUM ('default', 'liked', 'disliked');

CREATE TABLE swipes (
    id bigserial NOT NULL PRIMARY KEY,
    user_id integer NOT NULL,
    other_user_id integer NOT NULL,
    status status NOT NULL DEFAULT 'default',
    other_status status NOT NULL DEFAULT 'default',
    created_time timestamp,
    updated_time timestamp,
    UNIQUE (user_id, other_user_id)
);
```

#3 Sharding Principles

- Scalability
  - Starting with fewer servers
  - Scaling to more servers with less effort
- Performance
  - Nearby users filtering in real-time
  - Large amount of swipes
- Simplicity
  - Easy to understand and implement
  - Sharding by user id

#4 Sharding

Physical Shards (Databases)



Logical Shards (Schemas)

Object ID (bigint)

timestamp in milliseconds	logical shard ID	sequence
41 bits	13 bits	10 bits

Swipes Redundancy

```
INSERT INTO shard1.swipes (user_id, other_user_id, status) VALUES (100, 200, 'liked');

INSERT INTO shard2.swipes (user_id, other_user_id, other_status) VALUES (200, 100, 'liked');
```

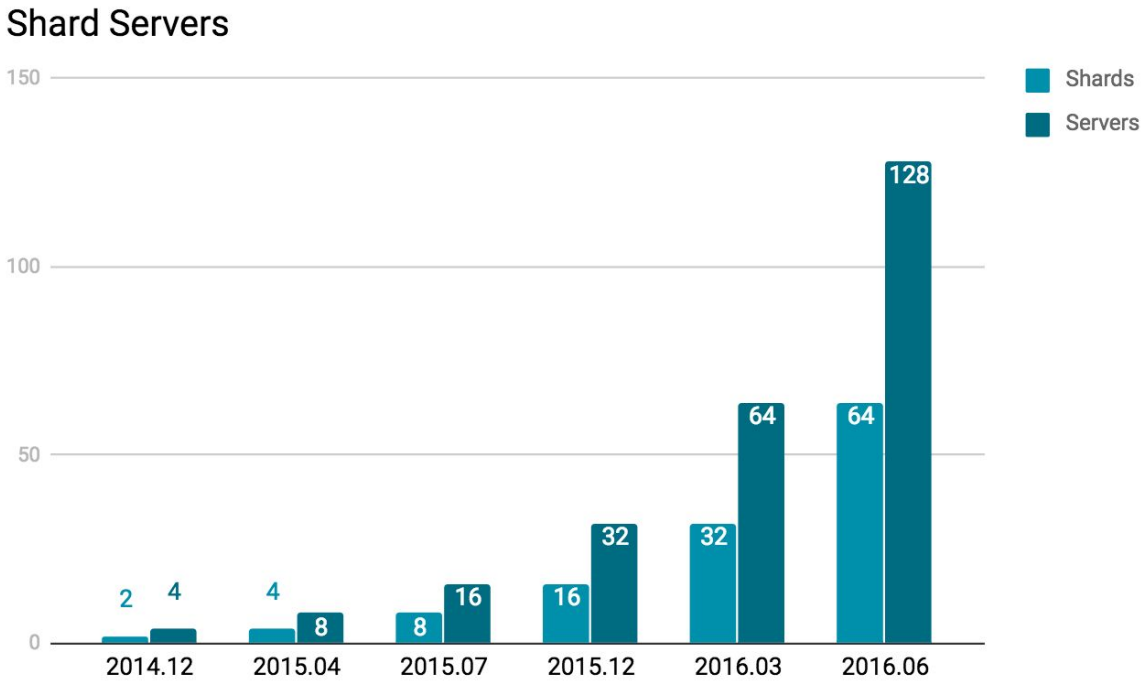
Reference: Sharding & ID at Instagram

#5 Sharding (continued)

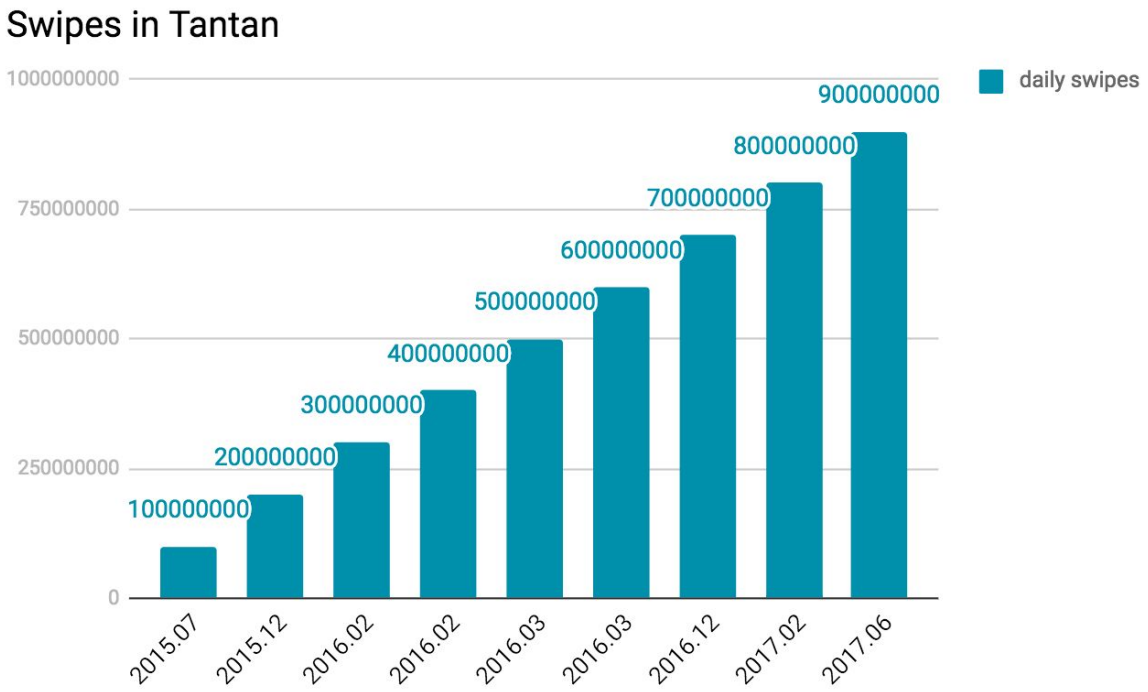
```
CREATE TYPE shard1.status AS ENUM ('default', 'liked', 'disliked');

CREATE TABLE shard1.swipes (
    id bigint NOT NULL DEFAULT shard1.swipe_id(),
    user_id integer NOT NULL,
    other_user_id integer NOT NULL,
    status shard1.status NOT NULL DEFAULT 'default',
    other_status shard1.status NOT NULL DEFAULT 'default',
    created_time timestamp,
    updated_time timestamp,
    UNIQUE (user_id, other_user_id)
);
```

```
CREATE SEQUENCE shard1.swipe_id_seq;

CREATE OR REPLACE FUNCTION shard1.swipe_id(OUT result bigint) AS
$$
DECLARE
    our_epoch bigint := 1314220021721;
    seq_id bigint;
    now_millis bigint;
    shard_id int := 1;
BEGIN
    SELECT nextval('shard1.swipe_id_seq') % 1024 INTO seq_id;

    SELECT FLOOR(EXTRACT(EPOCH FROM clock_timestamp()) * 1000) INTO now_millis;
    result := (now_millis - our_epoch) << 23;
    result := result | (shard_id <<10);
    result := result | (seq_id);
END;
$$ LANGUAGE PLPGSQL;
```

#6 Shards Split



#7 Swipes in Tantan



PostgreSQL in Tantan

#1 Overview

1M7S 1M1S 1M3S 1M1S x 64

#2 PostGiS

- GiST-based R-Tree spatial indexes
- Rich functions for analysis and processing of GIS objects
  - ST_Point, ST_Distance, ST_Contains etc.
- Scenarios
  - Find nearby users
  - Calculate distance between users
  - Construct regions database

#3 Partial indexes

- Avoid indexing common values
- Speed up update operations
- Scenarios
  - Find nearby users based on seach gender

```
CREATE INDEX ON users USING gist (location) WHERE location IS NOT NULL AND gender = 'female';

CREATE INDEX ON users USING gist (location) WHERE location IS NOT NULL AND gender = 'male' AND
(search_gender = ANY(ARRAY['both', 'male']));
```

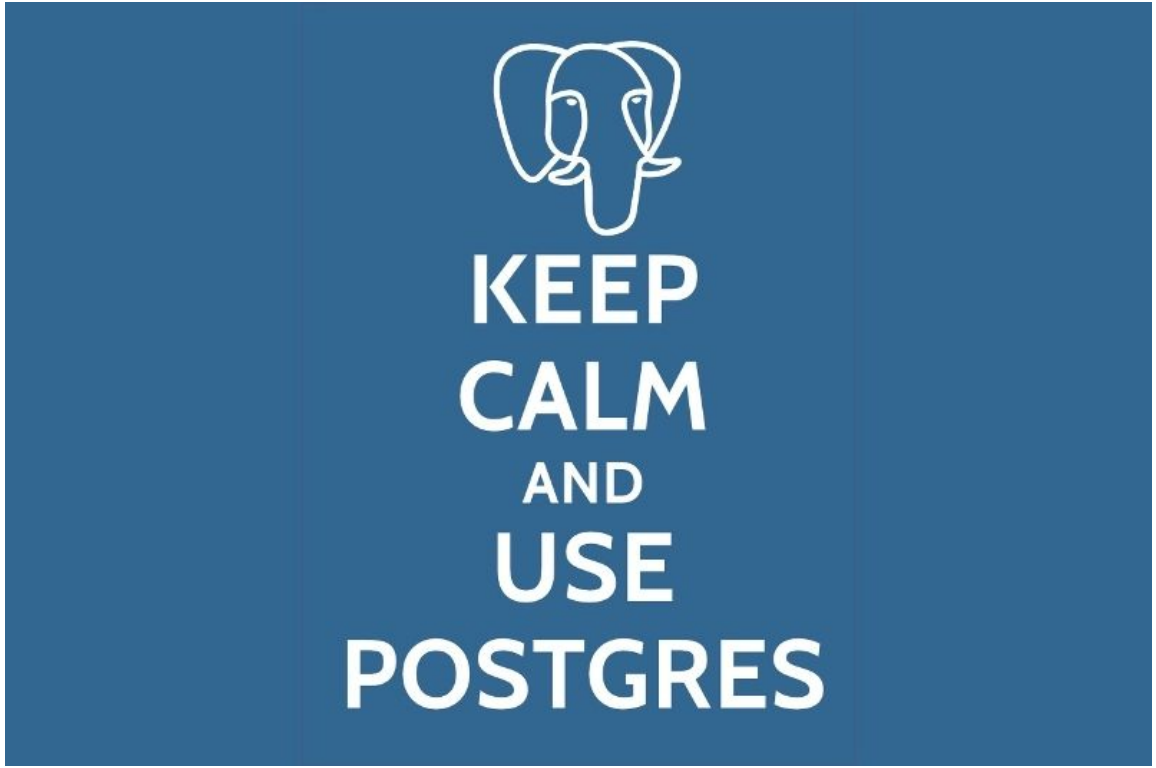
#4 Stored Procedures

- Separation of concerns
- Save extra round trips between client and server
- Flexibility in using PL/pgSQL
- Monitoring

```
CREATE OR REPLACE FUNCTION shard1.upsert_swipe(
    user_id_ integer,
    other_user_id_ integer,
    status_ shard1.status) RETURNS SETOF shard1.swipes AS
$$
BEGIN
    RETURN QUERY INSERT INTO shard1.swipes(
            user_id,
            other_user_id,
            status,
            created_time
        )
        VALUES
        (
            user_id_,
            other_user_id_,
            status_,
            CURRENT_TIMESTAMP AT TIME ZONE 'UTC'
        ) RETURNING *;
    EXCEPTION WHEN unique_violation THEN
        RETURN QUERY UPDATE shard1.swipes
            SET
                status = status_
            WHERE
                user_id = user_id_
            AND
                other_user_id = other_user_id_
            RETURNING *;
END;
$$ LANGUAGE PLPGSQL;
```

#5 No Downtime Operations

- Create or replace a function
- Add a new column
- Add a new non-nullable column with a default value (4 steps)
- Add a default value to an existing column
- Add an index concurrently
- Drop a column
- Drop a constraint
- etc.


KEEP
CALM
AND
USE
POSTGRES

Thanks!

Henry Ren