SCHEMA DESIGN MISTAKES

{How to not make some}
SCHEMA DESIGN

EYE-CATCHING
GOOD USE OF NEGATIVE SPACE
CONTRAST
ARRANGEMENT
EVOCATIVE OF THE STRUGGLE OF MODERN MAN
DDL: Data Definition Language

- CREATE TABLE ...
- CREATE INDEX ON...
- ALTER TABLE ...
- CREATE TYPE ...
- DROP TABLE ...
About me

- Open source software engineer
- Postgres fan
- Over decade of professional DB-driven app development
- Co-founder and CTO of DoctorBase
- Lived with my bad decisions
Topics

- Normalization
- Dumb schema design
- Good schema design
- Data integrity
- Pro tips
Normalization

- Eliminating data redundancy + prevent update anomalies
- Have a unique way to refer to rows for update/delete
- 1NF, 2NF, 3NF
DUMB SCHEMA

- Dumb table name
- Not normalized - speaker
- No good unique primary key
- Timezone-dependent

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>talk</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>speaker</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>timestamp without time zone</td>
<td>default now()</td>
</tr>
<tr>
<td>name</td>
<td>speaker</td>
<td>time</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>How to suck at RDBMSes</td>
<td>Mischa Spiegelmock</td>
<td>2016-01-25 23:46:49.659108</td>
</tr>
</tbody>
</table>

(2 rows)
CREATE TABLE speaker (  
id SERIAL PRIMARY KEY,  
created TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
person_name TEXT);

CREATE TABLE talk (  
id SERIAL PRIMARY KEY,  
created TIMESTAMPTZ NOT NULL DEFAULT NOW(),  
talk_title TEXT,  
speaker INTEGER NOT NULL REFERENCES speaker(id) ON DELETE CASCADE,  
time TIMESTAMPTZ NOT NULL);

ALTER TABLE talk ADD CONSTRAINT "table_unique_talk_speaker" UNIQUE (talk_title, speaker);
dumb=# INSERT INTO speaker (person_name) VALUES ('Mischa Spiegelmock');
INSERT 0 1

dumb=# INSERT INTO TALK (talk_title, speaker, time) VALUES ('Schema Design', 1, '2016-02-10');
INSERT 0 1

dumb=# SELECT * FROM TALK;
<table>
<thead>
<tr>
<th>id</th>
<th>created</th>
<th>talk_title</th>
<th>speaker</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-02-09 18:08:08.592791</td>
<td>Schema Design</td>
<td>1</td>
<td>2016-02-10 00:00:00</td>
</tr>
</tbody>
</table>
(1 row)

dumb=# DELETE FROM speaker WHERE id=1;
DELETE 1

dumb=# SELECT * FROM TALK;
<table>
<thead>
<tr>
<th>id</th>
<th>created</th>
<th>person_name</th>
<th>speaker</th>
<th>time</th>
</tr>
</thead>
</table>
(0 rows)
CREATE TABLE postgresql_talk (id SERIAL PRIMARY KEY,

• Composite PKs - DUMB
• Josh Berkus dislikes surrogate keys 😞
• External system PK - DUMB
• No PK - EXTRA DUMB
CREATE TABLE talk_review (  id SERIAL PRIMARY KEY,  created TIMESTAMPTZ NOT NULL DEFAULT NOW(),  talk_id INTEGER REFERENCES talk(id),  recommendable BOOLEAN,  email TEXT,  rating INTEGER);

ALTER TABLE talk_review ADD CONSTRAINT "review_valid_rating"  CHECK (rating IS NULL OR rating > 0 AND rating <= 5);
CREATE MATERIALIZED VIEW review_stat AS
SELECT
    talk_id,
    COUNT( * ) AS total_reviews,
    ROUND( AVG( rating ), 2 ) AS talk_rating,
    SUM( CASE WHEN email IS NOT NULL THEN 1 ELSE 0 END ) AS total_reviews_with_email,
    ROUND( AVG( recommendable ), 2 ) AS avg_recommendable,
FROM talk_review WHERE rating IS NOT NULL
GROUP BY talk_id ORDER BY talk_id;

REFRESH MATERIALIZED VIEW review_stat;
```sql
> INSERT INTO talk_review (talk_id, recommendable, email, rating) VALUES (1, 'f', NULL, 3);
> INSERT INTO talk_review (talk_id, recommendable, email, rating) VALUES (1, 't', 'me@foo', 1);
> INSERT INTO talk_review (talk_id, recommendable, email, rating) VALUES (1, 'f', NULL, 4);

> SELECT * FROM review_stat;

<table>
<thead>
<tr>
<th>talk_id</th>
<th>total_reviews</th>
<th>talk_rating</th>
<th>total_reviews_with_email</th>
<th>avg_recommendable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2.6666666666666667</td>
<td>1</td>
<td>0.33333333333333333333</td>
</tr>
</tbody>
</table>

(1 row)
```
### TIMEZONES

<table>
<thead>
<tr>
<th>Time Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>US/Pacific</td>
</tr>
</tbody>
</table>

```
dumb=# show timezone;
   TimeZone
----------
   US/Pacific
(1 row)

dumb=# insert into postgresql_talks values ('How to suck at RDBMSes', 'Mischa Spiegelmock');
dumb=# insert into postgresql_talks values ('Pg LISTEN/NOTIFY dopeness', 'Mischa Spiegelbokkers');
dumb=# select * from postgresql_talks;
            talk_title            |        speaker        |            time
--------------------------------+-----------------------+-----------------------
   How to suck at RDBMSes        | Mischa Spiegelmock    | 2016-01-25 23:46:49.659108
```

(2 rows)
dumb=# set timezone='US/Hawaii';

dumb=# INSERT INTO postgresql_talks VALUES ('Time zones suck', 'Mischa Spiegelmock');
INSERT 0 1

dumb=# SELECT * FROM postgresql_talks;

<table>
<thead>
<tr>
<th>talk_title</th>
<th>speaker</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to suck at RDBMSes</td>
<td>Mischa Spiegelmock</td>
<td>2016-01-25 23:46:49.659108</td>
</tr>
<tr>
<td>Time zones suck</td>
<td>Mischa Spiegelmock</td>
<td>2016-01-25 21:48:34.619602</td>
</tr>
</tbody>
</table>

(3 rows)
TIMEZONE DUMBNESS

- “I’ll just use the system timezone”
- “I can store a timezone with a GMT offset”
- “I can move Pg to another machine and everything will stay the same”
- “I can plant more potatoes with DST”
- “Daylight savings time is regular”
<table>
<thead>
<tr>
<th>Rule</th>
<th>NAME</th>
<th>FROM</th>
<th>TO</th>
<th>TYPE</th>
<th>IN</th>
<th>ON</th>
<th>AT</th>
<th>SAVE</th>
<th>LETTER/S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>US</td>
<td>1918</td>
<td>1919</td>
<td>-</td>
<td>Mar</td>
<td>lastSun</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1918</td>
<td>1919</td>
<td>-</td>
<td>Oct</td>
<td>lastSun</td>
<td>2:00</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1942</td>
<td>only</td>
<td>-</td>
<td>Feb</td>
<td>9</td>
<td>2:00</td>
<td>1:00</td>
<td>W # War</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1945</td>
<td>only</td>
<td>-</td>
<td>Aug</td>
<td>14</td>
<td>23:00u</td>
<td>1:00</td>
<td>P # Peace</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1945</td>
<td>only</td>
<td>-</td>
<td>Sep</td>
<td>30</td>
<td>2:00</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1967</td>
<td>2006</td>
<td>-</td>
<td>Oct</td>
<td>lastSun</td>
<td>2:00</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1967</td>
<td>1973</td>
<td>-</td>
<td>Apr</td>
<td>lastSun</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1974</td>
<td>only</td>
<td>-</td>
<td>Jan</td>
<td>6</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1975</td>
<td>only</td>
<td>-</td>
<td>Feb</td>
<td>23</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1976</td>
<td>1986</td>
<td>-</td>
<td>Apr</td>
<td>lastSun</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>1987</td>
<td>2006</td>
<td>-</td>
<td>Apr</td>
<td>Sun&gt;=1</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>2007</td>
<td>max</td>
<td>-</td>
<td>Mar</td>
<td>Sun&gt;=8</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>US</td>
<td>2007</td>
<td>max</td>
<td>-</td>
<td>Nov</td>
<td>Sun&gt;=1</td>
<td>2:00</td>
<td>0</td>
<td>S</td>
</tr>
</tbody>
</table>

....

<table>
<thead>
<tr>
<th>Rule</th>
<th>NAME</th>
<th>FROM</th>
<th>TO</th>
<th>TYPE</th>
<th>IN</th>
<th>ON</th>
<th>AT</th>
<th>SAVE</th>
<th>LETTER/S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>NYC</td>
<td>1920</td>
<td>only</td>
<td>-</td>
<td>Mar</td>
<td>lastSun</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>NYC</td>
<td>1920</td>
<td>only</td>
<td>-</td>
<td>Oct</td>
<td>lastSun</td>
<td>2:00</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>Rule</td>
<td>NYC</td>
<td>1921</td>
<td>1966</td>
<td>-</td>
<td>Apr</td>
<td>lastSun</td>
<td>2:00</td>
<td>1:00</td>
<td>D</td>
</tr>
<tr>
<td>Rule</td>
<td>NYC</td>
<td>1921</td>
<td>1954</td>
<td>-</td>
<td>Sep</td>
<td>lastSun</td>
<td>2:00</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>Rule</td>
<td>NYC</td>
<td>1955</td>
<td>1966</td>
<td>-</td>
<td>Oct</td>
<td>lastSun</td>
<td>2:00</td>
<td>0</td>
<td>S</td>
</tr>
</tbody>
</table>

# Zone NAME GMTOFF RULES FORMAT [UNTIL]
Zone America/New_York -4:56:02 LMT 1883 Nov 18 12:03:58
-5:00 US E%st 1920
-5:00 NYC E%st 1942
-5:00 US E%st 1946
-5:00 NYC E%st 1967
-5:00 US E%st

Maintainer: Paul Eggert
Sane Dates and Times

- Store everything in GMT or Unix Timestamp
- Use type: TIMESTAMPTZ
- Convert to local time on display
- Store TZ as IANA name (“America/Los_Angeles”)
- E-Z date math (1 day = 86400, 1 hr = 3600, etc)
- Never store age, store birthdate
CREATE TABLE postgresql_talk (id SERIAL PRIMARY KEY, created TIMESTAMPTZ NOT NULL DEFAULT NOW())

• Reporting
• Fix blunders in a date range
ALTER TABLE foo ADD CONSTRAINT "must_choose_one" CHECK ((A is null)::int + (B is null)::int + (C is null)::int = 2);

CREATE TYPE foo_opt AS ENUM ('A', 'B', 'C');

CREATE TYPE sex AS ENUM ('male', 'female', 'other');
TRIGGER WARNING: TRIGGERS
CREATE FUNCTION user_keyword_update() RETURNS trigger LANGUAGE plpgsql AS $$
BEGIN
    NEW.keywords := to_tsvector('pg_catalog.english',
        NEW.id || ' ' ||
        coalesce(NEW.username,'') || ' ' ||
        coalesce(NEW.email_address,'') || ' ' ||
        coalesce(NEW.name,'') || ' ' ||
        coalesce(NEW.phone,'') || ' ' ||
    );
    RETURN NEW;
END;
$$;

CREATE TRIGGER user_keyword_update_trigger BEFORE INSERT OR UPDATE ON "user" FOR EACH ROW EXECUTE PROCEDURE user_keyword_update();
CREATE OR REPLACE FUNCTION update_georeferenced_table() RETURNS TRIGGER AS $$
DECLARE
    name TEXT;
BEGIN
    name := TG_ARGV[0];
    IF ((TG_OP = 'INSERT' AND NEW.location IS NOT NULL) OR (TG_OP = 'UPDATE' AND NEW.location IS DISTINCT FROM OLD.location)) THEN
        NEW.updated_location := current_timestamp;
    END IF;
    PERFORM pg_notify(name || '_updated', ' {
        "id": ' || CAST(NEW.id AS TEXT) || ', "location": ' || ST_AsGeoJSON(NEW.location) || ' }
    ');
RETURN NEW;
END;
$$
LISTEN/NOTIFY \[\leftrightarrow\] WebSocket

github.com/doctorbase/wsnotify

github.com/revmischa/pgnotify-demos
ULTIMATE KILLER SCHEMA

“I’m going to make the ultimo super flexible schema and let my users choose what fields to store”

CREATE TABLE user_field (  
    user_id integer NOT NULL,  
    field_id integer NOT NULL,  
    value text  
);

CREATE TABLE field (  
    id integer SERIAL PRIMARY KEY,  
    name text  
);
Key-value pair store: JSONB

Join the No-noSQL movement
Object Relational Mappers

- Define classes for tables
- Instance == row
- Hibernate, DBIx::Class, Django-ORM, SQLAlchemy

```sql
SELECT * FROM talk t LEFT OUTER JOIN talk_review tr ON tr.talk_id=t.id WHERE t.id=1;
```

```perl
my @reviews = $talk->reviews;
```
```python
query = db.session
    .query(func.count(UserTeam.team_id))
    .filter(UserTeam.user_id.in_([ self.id, user.id ]))
    .group_by(UserTeam.team_id)
    .having(func.count(UserTeam.user_id) > 1)
```
class PointGeography(types.UserDefinedType):
    def get_col_spec(self):
        return "GEOMETRY"
    def column_expression(self, col):
        return ST_AsGeoJSON(col, type_=self)

class GeoReferenced():
    updated_location = Column(DateTime())
    location_accuracy_meters = Column(Numeric())
    location = Column(PointGeography)
    def set_location(self, lng, lat):
        if lat is None or lng is None:
            self.location = None
        else:
            self.location = "POINT(%0.16f %0.16f)" %
            (float(lng), float(lat))
    db.session.commit()

    @property
    def lat(self):
        if self.location is None:
            return None
        return parse_point(self.location)[1]

    @property
    def lng(self):
        if self.location is None:
            return None
        return parse_point(self.location)[0]

def parse_point(point):
    geo = json.loads(point)
    return geo['coordinates']

@classmethod
def within_clause(cls, latitude, longitude, distance):
    attr = '%s.location' % cls.__tablename__
    point = 'POINT(%0.8f %0.8f)' % (longitude, latitude)
    location = "ST_GeographyFromText(E'SRID=4326;%s')" % point
    return 'ST_DWithin(%s, %s, %d)' % (attr, location, distance)
SET statement_timeout = 120000
BLOBS / BYTEA

Never worked well for me.
Use TEXT

VARCHAR(NN) is pointless
No plurals!
| USER TABLE |

Don’t call it “user”
Lots of foreign keys with no indexes == slow DELETE
Make schemas for related data
Schema should map real things
WHEN TO USE STORED PROCEDURES?

Thoughts?
Storing Static Files

- Amazon S3
- Security tokens
- Static web hosting
- Fancy ACLs
- Requires application-level support
- Store s3keyname column (and MIME, size, views, s3bucket, …)
HELP!

irc.freenode.net #postgresql

Slack: https://denish1.typeform.com/to/S7h9jI