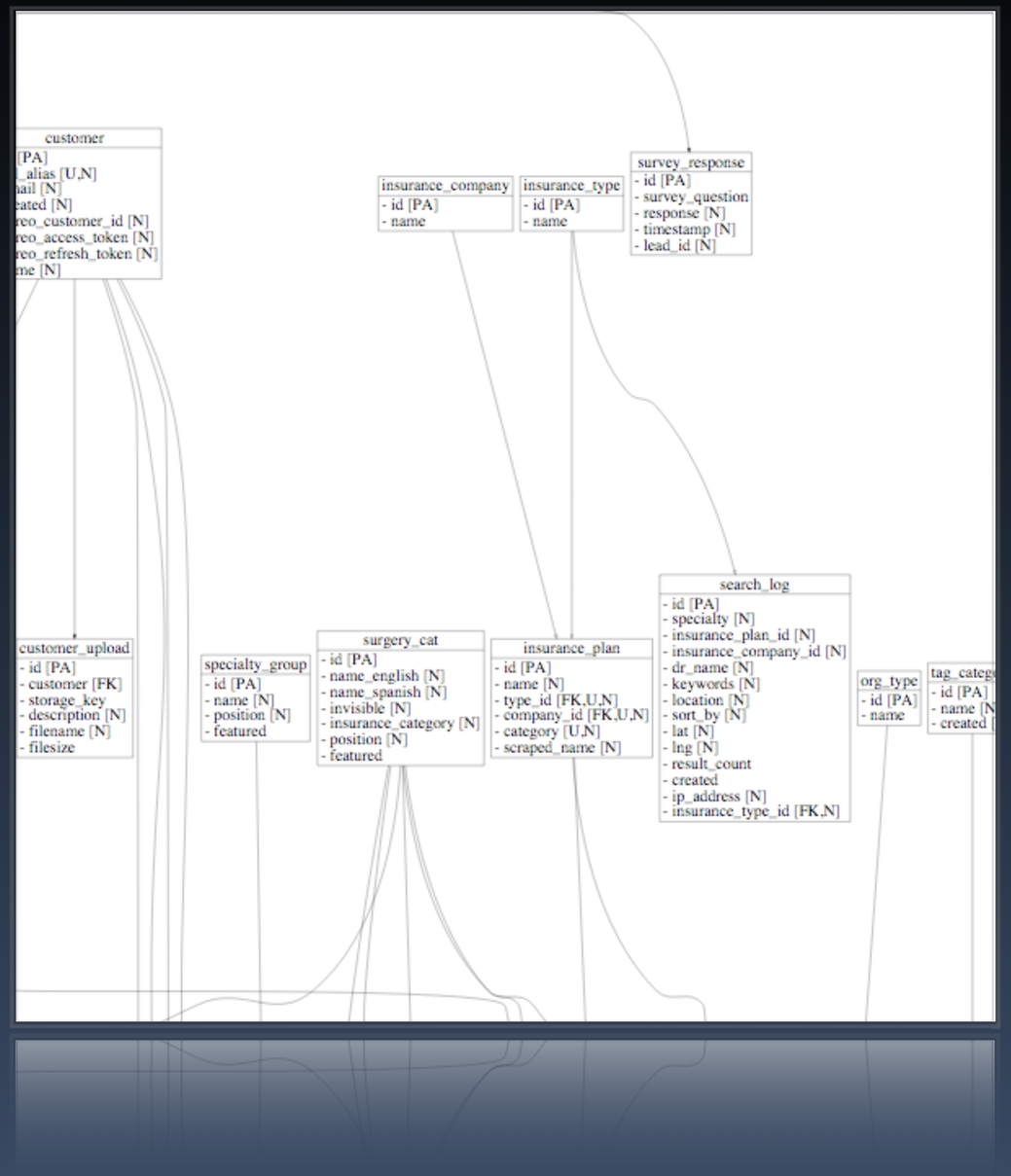
SCHEMA DESIGN MISTAKES

{How to not make some}

SCHEMA DESIGN

EYE-CATCHING
GOOD USE OF NEGATIVE SPACE
CONTRAST
ARRANGEMENT
EVOCATIVE OF THE STRUGGLE OF MODERN MAN



LESSONS IN SCHEMA DESIGN

LEARN FROM MY MISTAKES

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
- Great with PostgreSQL foreign keys

DUMB SCHEMA

```
            Table "postgresql_talks"
 Column  |            Type                |   Modifiers
---------+--------------------------------+---------------
 talk    | text                           |
 speaker | text                           |
 time    | timestamp without time zone | default now()
```

- **Dumb table name**
- **No primary key ID**
- **Speaker should be foreign key**
- **Timezone-dependent**

🔑 KEYS

```
CREATE TABLE postgresql_talk (
 id SERIAL PRIMARY KEY,
```

- Composite PKs - DUMB
- External system PK - DUMB
- No PK - EXTRA DUMB
- Lots of FKs - SLOW DELETES
- Josh Berkus dislikes surrogate keys 😟

DUMB SCHEMA

```
dumb=# insert into postgresql_talks values ('How to suck at RDBMSes', 'Mischa
Spiegelmock');

dumb=# insert into postgresql_talks values ('Pg LISTEN/NOTIFY dopeness', ‘Misha
Spiegelmock’);

dumb=# select * from postgresql_talks;
           name             |        speaker         |            time
----------------------------+------------------------+----------------------------
 How to suck at RDBMSes     | Mischa Spiegelmock     | 2016-01-25 23:46:49.659108
 Pg LISTEN/NOTIFY dopeness | Misha  Spiegelmock     | 2016-01-25 23:47:14.300665
(2 rows)
```

LESS DUMB SCHEMA

```
CREATE TABLE speaker (
    id SERIAL PRIMARY KEY,
    created TIMESTAMP NOT NULL DEFAULT NOW(),
    name TEXT
);

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

ALTER TABLE talk ADD CONSTRAINT "table_unique_talk_speaker" UNIQUE
(name, speaker);
```

LESS DUMB SCHEMA

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

dumb=# SELECT * FROM TALK;
 id |          created          |     name      | speaker |        time
----+---------------------------+---------------+---------+---------------------
  1 | 2016-02-09 18:08:08.592791 | Schema Design |       1 | 2016-02-10 00:00:00
(1 row)

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

dumb=# SELECT * FROM TALK;
 id | created | name | speaker | time
----+---------+------+---------+------
(0 rows)
```

TALK REVIEWS

```
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);
```

MATERIALIZED VIEW REPORTS

```
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;
```

REFRESH MATERIALIZED VIEW

```
> 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;
 talk_id | total_reviews | talk_rating | total_reviews_with_email | avg_recommendable
---------+---------------+-------------+--------------------------+-------------------
(0 rows)


> REFRESH MATERIALIZED VIEW review_stat;
REFRESH MATERIALIZED VIEW
> SELECT * FROM review_stat;
 talk_id | total_reviews |    talk_rating     | total_reviews_with_email |   avg_recommendable
---------+---------------+--------------------+--------------------------+-----------------------
       1 |             3 | 2.6666666666666667 |                        1 | 0.3333333333333333
(1 row)
```

TIMEZONES

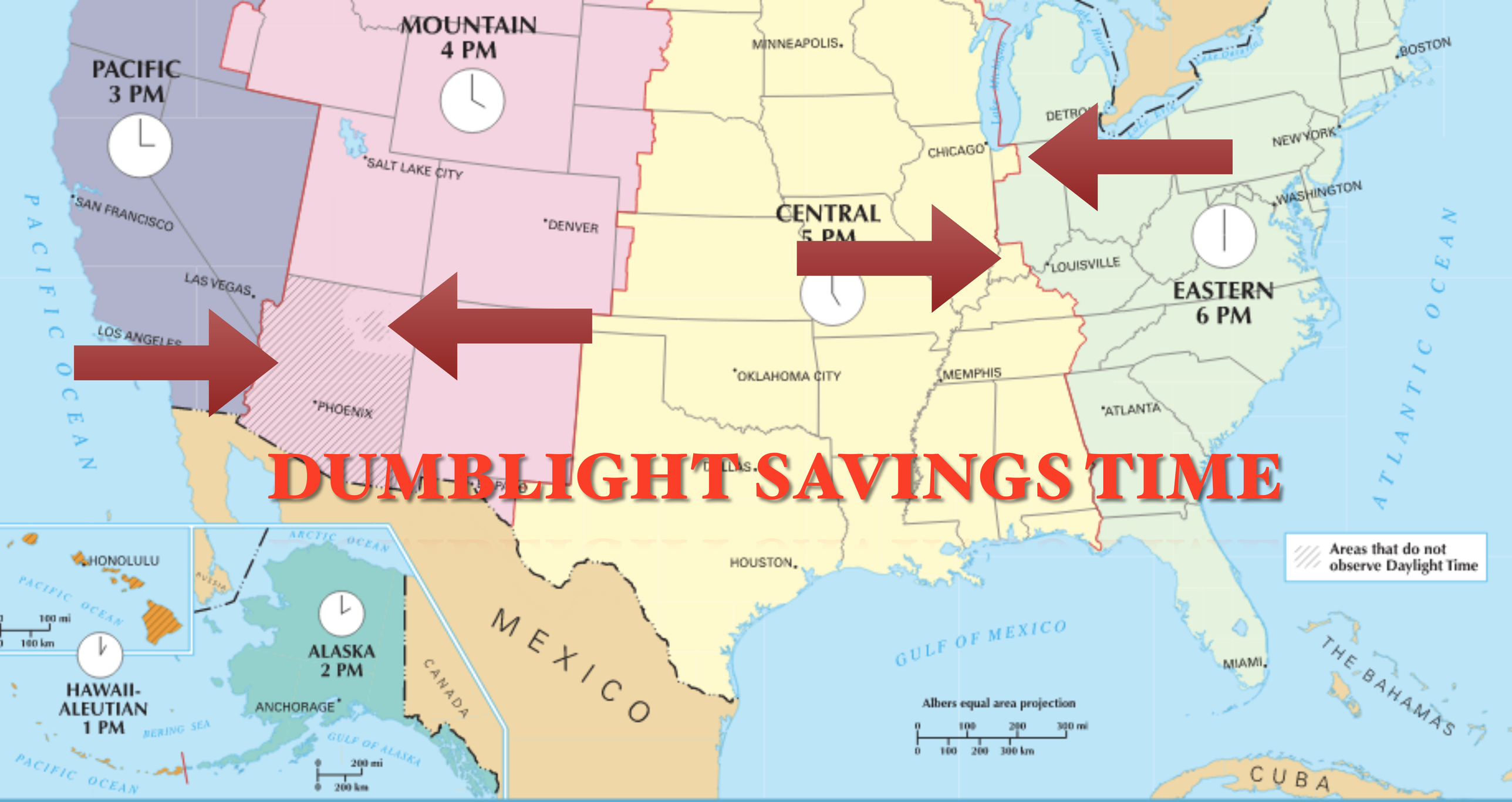
```
dumb=# show timezone;
  TimeZone
------------
 US/Pacific
(1 row)

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;
           name             |        speaker         |            time
----------------------------+------------------------+----------------------------
 How to suck at RDBMSes     | Mischa Spiegelmock     | 2016-01-25 23:46:49.659108
 Pg LISTEN/NOTIFY dopeness | Mischa Spiegelbokkers | 2016-01-25 23:47:14.300665
(2 rows)
```

TIMEZONES

```
dumb=# set timezone='US/Hawaii';
dumb=# insert into postgresql_talks values ('Time zones suck', 'Mischa
Spiegelmock');
INSERT 0 1
dumb=# select * from postgresql_talks;
           name              |        speaker         |            time
-----------------------------+------------------------+----------------------------
 How to suck at RDBMSes      | Mischa Spiegelmock     | 2016-01-25 23:46:49.659108
 Pg LISTEN/NOTIFY dopeness   | Mischa Spiegelbokkers  | 2016-01-25 23:47:14.300665
 Time zones suck             | Mischa Spiegelmock     | 2016-01-25 21:48:34.619602
(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”



DUMBLIGHT SAVINGS TIME

U.S. Department of the Interior
U.S. Geological Survey

The National Atlas of the United States of America®

```
# Rule  NAME     FROM     TO       TYPE     IN       ON       AT       SAVE     LETTER/S
Rule    US       1918     1919     -        Mar      lastSun  2:00     1:00     D
Rule    US       1918     1919     -        Oct      lastSun  2:00     0        S
Rule    US       1942     only     -        Feb      9        2:00     1:00     W # War
Rule    US       1945     only     -        Aug      14       23:00u   1:00     P # Peace
Rule    US       1945     only     -        Sep      30       2:00     0        S
Rule    US       1967     2006     -        Oct      lastSun  2:00     0        S
Rule    US       1967     1973     -        Apr      lastSun  2:00     1:00     D
Rule    US       1974     only     -        Jan      6        2:00     1:00     D
Rule    US       1975     only     -        Feb      23       2:00     1:00     D
Rule    US       1976     1986     -        Apr      lastSun  2:00     1:00     D
Rule    US       1987     2006     -        Apr      Sun>=1   2:00     1:00     D
Rule    US       2007     max      -        Mar      Sun>=8   2:00     1:00     D
Rule    US       2007     max      -        Nov      Sun>=1   2:00     0        S
....
# Rule  NAME     FROM     TO       TYPE     IN       ON       AT       SAVE     LETTER
Rule    NYC      1920     only     -        Mar      lastSun  2:00     1:00     D
Rule    NYC      1920     only     -        Oct      lastSun  2:00     0        S
Rule    NYC      1921     1966     -        Apr      lastSun  2:00     1:00     D
Rule    NYC      1921     1954     -        Sep      lastSun  2:00     0        S
Rule    NYC      1955     1966     -        Oct      lastSun  2:00     0        S
# 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

<END TIMEZONE RANT>

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 (1day = 86400, 1hr = 3600, etc)
- Never store age, store birthdate

.CREATED

```
CREATE TABLE postgresql_talk (
 id SERIAL PRIMARY KEY,
 created TIMESTAMPTZ NOT NULL DEFAULT NOW()
```

- Reporting
- Fix blunders in a date range

CONSTRAINTS

```
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

TSVECTOR 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();
```

LOCATION UPDATE TRIGGERS

```
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 ⟷ 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
);
```

EAV

Key-value pair store: JSONB

Join the No-noSQL movement

Example	Example Result
`to_json('Fred said "Hi."'::text)`	`"Fred said \"Hi.\""`
`array_to_json('{{1,5}, {99,100}}'::int[])`	`[[1,5],[99,100]]`
`row_to_json(row(1,'foo'))`	`{"f1":1,"f2":"foo"}`
`json_build_array(1,2,'3',4,5)`	`[1, 2, "3", 4, 5]`
`json_build_object('foo',1,'bar',2)`	`{"foo": 1, "bar": 2}`

`json_object('{a, 1, b, "def", c, 3.5}')`
 	
`json_object('{{a, 1},{b, "def"}, {c, 3.5}}')`
	`{"a": "1", "b": "def", "c": "3.5"}`

`json_object('{a, b}', '{1,2}')`	`{"a": "1", "b": "2"}`

Object Relational Mappers

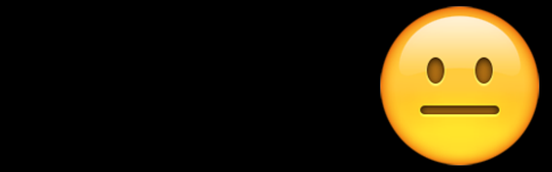
- Define classes for tables
- Instance == row
- Hibernate, DBIx::Class, Django-ORM, SQLAlchemy

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

```
my @reviews = $talk->reviews;
```

ORM

```
select team_id from user_team
  where user_id in ($self.id, $user.id)
  group by team_id
  having count(*) > 1;

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)
```



ORM - POSTGIS

```
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)
```

TIMEOUT

```
SET statement_timeout = 120000
```

BLOBS / BYTEA

Never worked well for me.

VARCHAR(255)

Use TEXT

VARCHAR(NN) is pointless

NAMING

No plurals!

USER TABLE

Don’t call it “user”

COMPARTMENTALIZATION

Make schemas for related data

MODEL DATA ON THE REAL WORLD

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/S7h9jl