Data Modeling, Normalization and Denormalisation

Dimitri Fontaine
Citus Data, now part of Microsoft
@tapoueh
The Art of PostgreSQL
Turn Thousands of Lines of Code into Simple Queries
PostgreSQL Major Contributor
CURRENTLY WORKING AT

Citus Data
Join us!

pg_auto_failover
Automated Failover

*PostgreSQL Licence, GitHub, fully open*
Migrating to PostgreSQL

In a single command line!
One-command migration

$ pgloader mysql://root@localhost/f1db?useSSL=false \ pgsql://f1db@localhost/f1db
```
$ pgloader ./test/mysql/f1db.load
2019-06-19T11:24:36.014000+02:00 LOG pgloader version "3.6.26cc9ca"
2019-06-19T11:24:36.154000+02:00 LOG Migrating from #<MYSQL-CONNECTION mysql://root@localhost:3306/f1db {100620ACC3}>
2019-06-19T11:24:36.155000+02:00 LOG Migrating into #<PGSQL-CONNECTION pgsql://dim@UNIX:5432/plop {100620B583}>
2019-06-19T11:24:41.001000+02:00 LOG report summary reset

<table>
<thead>
<tr>
<th>table name</th>
<th>errors</th>
<th>rows</th>
<th>bytes</th>
<th>total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch meta data</td>
<td>0</td>
<td>33</td>
<td></td>
<td>0.413s</td>
</tr>
<tr>
<td>Create Schemas</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.002s</td>
</tr>
<tr>
<td>Create SQL Types</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.005s</td>
</tr>
<tr>
<td>Create tables</td>
<td>0</td>
<td>26</td>
<td></td>
<td>0.174s</td>
</tr>
<tr>
<td>Set Table OIDs</td>
<td>0</td>
<td>13</td>
<td></td>
<td>0.007s</td>
</tr>
<tr>
<td>f1db.circuits</td>
<td>0</td>
<td>73</td>
<td>8.5 kB</td>
<td>0.024s</td>
</tr>
<tr>
<td>f1db.constructorresults</td>
<td>0</td>
<td>11142</td>
<td>186.2 kB</td>
<td>0.089s</td>
</tr>
<tr>
<td>f1db.constructors</td>
<td>0</td>
<td>208</td>
<td>15.0 kB</td>
<td>0.113s</td>
</tr>
<tr>
<td>f1db.constructorstandings</td>
<td>0</td>
<td>11896</td>
<td>249.3 kB</td>
<td>0.242s</td>
</tr>
<tr>
<td>f1db.drivers</td>
<td>0</td>
<td>842</td>
<td>79.8 kB</td>
<td>0.175s</td>
</tr>
<tr>
<td>f1db.laptimes</td>
<td>0</td>
<td>42663</td>
<td>11.2 MB</td>
<td>2.148s</td>
</tr>
<tr>
<td>f1db.driverstandings</td>
<td>0</td>
<td>31726</td>
<td>719.1 kB</td>
<td>0.456s</td>
</tr>
<tr>
<td>f1db.pitstops</td>
<td>0</td>
<td>6251</td>
<td>209.6 kB</td>
<td>0.351s</td>
</tr>
<tr>
<td>f1db.races</td>
<td>0</td>
<td>997</td>
<td>100.6 kB</td>
<td>0.353s</td>
</tr>
<tr>
<td>f1db.seasons</td>
<td>0</td>
<td>69</td>
<td>3.9 kB</td>
<td>0.384s</td>
</tr>
<tr>
<td>f1db.qualifying</td>
<td>0</td>
<td>7516</td>
<td>286.4 kB</td>
<td>0.094s</td>
</tr>
<tr>
<td>f1db.results</td>
<td>0</td>
<td>23777</td>
<td>1.3 MB</td>
<td>0.276s</td>
</tr>
<tr>
<td>f1db.status</td>
<td>0</td>
<td>134</td>
<td>1.7 kB</td>
<td>0.023s</td>
</tr>
</tbody>
</table>

COPY Threads Completion | 0  | 4  | 2.549s  |
Create Indexes         | 0  | 20 | 2.396s  |
Index Build Completion | 0  | 20 | 1.322s  |
Reset Sequences        | 0  | 10 | 0.105s  |
Primary Keys           | 0  | 13 | 0.020s  |
Create Foreign Keys    | 0  | 0  | 0.000s  |
Create Triggers        | 0  | 0  | 0.001s  |
Set Search Path        | 0  | 1  | 0.001s  |
Install Comments       | 0  | 0  | 0.000s  |

Total import time ✓ 521264 14.3 MB 6.394s
```
The Art of PostgreSQL

Turn Thousands of Lines of Code into Simple Queries

by Dimitri Fontaine
Data Modeling
Rule 5. Data dominates.

“If you’ve chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.”

(Brooks p. 102)
Data Modeling Examples

- Data Types
- Constraints
- Primary keys, Foreign Keys, Check, Not Null
- Partial unique indexes
- Exclusion Constraints
create table sandbox.article
(
    id         bigserial primary key,
    category   integer references sandbox.category(id),
    pubdate    timestamptz,
    title      text not null,
    content    text
);

Data Modeling
CREATE TABLE toggles
(
    user_id    integer    NOT NULL,
    type       text       NOT NULL,
    enabled_at timestamp NOT NULL,
    disabled_at timestamp,
);

CREATE UNIQUE INDEX ON toggles (user_id, type)
    WHERE disabled_at IS NULL;
Constraints are Guarantees

create table rates
(
    currency text,
    validity daterange,
    rate numeric,

    exclude using gist (currency with =,
                          validity with &&)
);

Avoiding Database Anomalies
## Update Anomaly

### Employees' Skills

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Employee Address</th>
<th>Skill</th>
</tr>
</thead>
<tbody>
<tr>
<td>426</td>
<td>87 Sycamore Grove</td>
<td>Typing</td>
</tr>
<tr>
<td>426</td>
<td>87 Sycamore Grove</td>
<td>Shorthand</td>
</tr>
<tr>
<td>519</td>
<td>94 Chestnut Street</td>
<td>Public Speaking</td>
</tr>
<tr>
<td>519</td>
<td>96 Walnut Avenue</td>
<td>Carpentry</td>
</tr>
</tbody>
</table>
## Insertion Anomaly

### Faculty and Their Courses

<table>
<thead>
<tr>
<th>Faculty ID</th>
<th>Faculty Name</th>
<th>Faculty Hire Date</th>
<th>Course Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>389</td>
<td>Dr. Giddens</td>
<td>10-Feb-1985</td>
<td>ENG-206</td>
</tr>
<tr>
<td>407</td>
<td>Dr. Saperstein</td>
<td>19-Apr-1999</td>
<td>CMP-101</td>
</tr>
<tr>
<td>407</td>
<td>Dr. Saperstein</td>
<td>19-Apr-1999</td>
<td>CMP-201</td>
</tr>
<tr>
<td>424</td>
<td>Dr. Newsome</td>
<td>29-Mar-2007</td>
<td></td>
</tr>
</tbody>
</table>
## Deletion anomaly

**Faculty and Their Courses**

<table>
<thead>
<tr>
<th>Faculty ID</th>
<th>Faculty Name</th>
<th>Faculty Hire Date</th>
<th>Course Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>389</td>
<td>Dr. Giddens</td>
<td>10-Feb-1985</td>
<td>ENG-206</td>
</tr>
<tr>
<td>407</td>
<td>Dr. Saperstein</td>
<td>19-Apr-1999</td>
<td>CMP-101</td>
</tr>
<tr>
<td>407</td>
<td>Dr. Saperstein</td>
<td>19-Apr-1999</td>
<td>CMP-201</td>
</tr>
</tbody>
</table>
ANOTHER QUOTE FROM FRED BROOKS

Database Design and User Workflow

“Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won’t usually need your flowcharts; they’ll be obvious.”
BEGIN;

create schema if not exists sandbox;

create table sandbox.category
(
    id    serial primary key,
    name  text not null
);

insert into sandbox.category(name)
    values ('sport'),('news'),('box office'),('music');

ROLLBACK;
Object Relational Mapping

• The **R** in ORM stands for relation

• Every SQL query result set is a relation
Object Relational Mapping

When mapping base tables, you end up trying to solve different complex issues at the same time

- User Workflow
- Consistent view of the whole world at all time
Normalization
Basics of the Unix Philosophy: principles

Clarity

- Clarity is better than cleverness

Simplicity

- Design for simplicity; add complexity only where you must.

Transparency

- Design for visibility to make inspection and debugging easier.

Robustness

- Robustness is the child of transparency and simplicity.
1st Normal Form, Codd, 1970

- There are no duplicated rows in the table.
- Each cell is single-valued (no repeating groups or arrays).
- Entries in a column (field) are of the same kind.
2nd Normal Form, Codd, 1971

“A table is in 2NF if it is in 1NF and if it has no partial dependencies.”

“A table is in 2NF if it is in 1NF and if all non-key attributes are dependent on all of the key. A partial dependency occurs when a non-key attribute is dependent on only a part of the composite key.”
Third Normal Form, Codd, 1971

BCNF, Boyce-Codd, 1974

• A table is in 3NF if it is in 2NF and if it has no transitive dependencies.

• A table is in BCNF if it is in 3NF and if every determinant is a candidate key.
More Normal Forms

• Each level builds on the previous one.

• A table is in 4NF if it is in BCNF and if it has no multi-valued dependencies.

• A table is in 5NF, also called “Projection-join Normal Form” (PJNF), if it is in 4NF and if every join dependency in the table is a consequence of the candidate keys of the table.

• A table is in DKNF if every constraint on the table is a logical consequence of the definition of keys and domains.
Database Constraints
Primary Keys

create table sandbox.article
(
    id bigserial primary key,
    category integer references sandbox.category(id),
    pubdate timestamptz,
    title text not null,
    content text
);

Artificially generated key is named a surrogate key because it is a substitute for natural key. A natural key would allow preventing duplicate entries in our data set.
Surrogate Keys

```
insert into sandbox.article
  (category, pubdate, title)
values (2, now(), 'Hot from the Press'),
       (2, now(), 'Hot from the Press')
returning *
```
Oops. Not a Primary Key.

<table>
<thead>
<tr>
<th>id</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>2</td>
</tr>
<tr>
<td>pubdate</td>
<td>2018-03-12 15:15:02.384105+01</td>
</tr>
<tr>
<td>title</td>
<td>Hot from the Press</td>
</tr>
<tr>
<td>content</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>2</td>
</tr>
<tr>
<td>pubdate</td>
<td>2018-03-12 15:15:02.384105+01</td>
</tr>
<tr>
<td>title</td>
<td>Hot from the Press</td>
</tr>
<tr>
<td>content</td>
<td></td>
</tr>
</tbody>
</table>

INSERT 0 2
create table sandboxpk.article
(
    category integer references sandbox.category(id),
    pubdate timestamptz,
    title text not null,
    content text,

    primary key(category, pubdate, title)
);

Natural Primary Key
create table sandboxpk.comment
(
    a_category integer     not null,
a_pubdate  timestamptz not null,
a_title    text        not null,
pubdate    timestamptz,
    content    text,

    primary key(a_category, a_pubdate, a_title, pubdate, content),

    foreign key(a_category, a_pubdate, a_title)
        references sandboxpk.article(category, pubdate, title)
);

Update Foreign Keys
create table sandbox.article
(
    id integer generated always as identity,
    category integer not null references sandbox.category(id),
    pubdate timestamptz not null,
    title text not null,
    content text,

    primary key(category, pubdate, title),
    unique(id)
);

Natural and Surrogate Keys
Other Constraints
Normalisation Helpers

- **Primary Keys**
- **Foreign Keys**
- **Not Null**
- **Check Constraints**
- **Domains**
- **Exclusion Constraints**

```sql
create table rates
(
    currency text,
    validity daterange,
    rate numeric,
    exclude using gist
    (currency with =,
    validity with &&
    )
);
```
Denormalization
Rules of Optimization

Rule 1: Don't do it.
Rule 2: Don't do it yet (experts only)
Premature Optimization...

“Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.”

Denormalization: cache

- Duplicate data for faster access
- Implement cache invalidation
Denormalization example

\set season 2017

select drivers.surname as driver, 
    constructors.name as constructor, 
    sum(points) as points

from results
    join races using(raceid)
    join drivers using(driverid)
    join constructors using(constructorid)

where races.year = :season

group by grouping sets(drivers.surname, constructors.name)
    having sum(points) > 150
order by drivers.surname is not null, points desc;
Denormalization example

create view v.season_points as
  select year as season, driver, constructor, points
  from seasons left join lateral
    (select drivers.surname as driver,
       constructors.name as constructor,
       sum(points) as points
    from results
    join races using(raceid)
    join drivers using(driverid)
    join constructors using(constructorid)
    where races.year = seasons.year
    group by grouping sets(drivers.surname, constructors.name)
    order by drivers.surname is not null, points desc
  ) as points on true
order by year, driver is null, points desc;
create materialized view cache.season_points as
  select * from v.season_points;
create index on cache.season_points(season);
Materialized View

refresh materialized view cache.season_points;
select driver, constructor, points
from cache.season_points
where season = 2017
and points > 150;
Denormalization: audit trails

- Foreign key references to other tables won't be possible when those reference change and you want to keep a history that, by definition, doesn't change.

- The schema of your main table evolves and the history table shouldn’t rewrite the history for rows already written.
History tables with JSONB

create schema if not exists archive;

create type archive.action_t
    as enum('insert', 'update', 'delete');

create table archive.older_versions
  (  
    table_name text,  
    date       timestamptz default now(),  
    action     archive.action_t,  
    data       jsonb  
  );
create table rates
(
    currency text,
    validity daterange,
    rate numeric,

    exclude using gist (currency with =,
                          validity with &&)
);

Validity Periods
Validity Periods

```
select currency, validity, rate
from rates
where currency = 'Euro'
  and validity @> date '2017-05-18';
```

- [ RECORD 1 ]---------------------
currency  | Euro
validity   | [2017-05-18,2017-05-19)
rate       | 1.240740
Denormalization Helpers: Data Types
Composite Data Types

- Composite Type
- Arrays
- JSONB
- Enum
- Domains

- hstore
- ltree
- intarray
- hll
Partitioning
Partitioning Improvements

PostgreSQL 10

- Indexing
- Primary Keys
- On conflict
- Update Keys

PostgreSQL 11

- Indexing, Primary Keys, Foreign Keys
- Hash partitioning
- Default partition
- On conflict support
- Update Keys
Schemaless with JSONB

```sql
select jsonb_pretty(data)
  from magic.cards
where data @> '{"type":"Enchantment",
    "artist":"Jim Murray",
    "colors":["Blue"]
}';
```
Durability Trade-Offs

create role dbowner with login;
create role app with login;

create role critical  with login in role app inherit;
create role notsomuch with login in role app inherit;
create role dontcare  with login in role app inherit;

alter user critical  set synchronous_commit to remote_apply;
alter user notsomuch set synchronous_commit to local;
alter user dontcare  set synchronous_commit to off;
Per Transaction Durability

SET demo.threshold TO 1000;

CREATE OR REPLACE FUNCTION public.syncrep_important_delta()
    RETURNS TRIGGER
    LANGUAGE PLpgSQL
    AS
    $$
    DECLARE
        threshold integer := current_setting('demo.threshold')::int;
        delta integer := NEW.abalance - OLD.abalance;
    BEGIN
        IF delta > threshold
            THEN
                SET LOCAL synchronous_commit TO on;
        END IF;
    END;
    $$;
Horizontal Scaling

Sharding with Citus
Five Sharding Data Models and which is right?

- Sharding by Geography
- Sharding by EntityId
- Sharding a graph
- Time Partitioning
The Art of PostgreSQL

Turn Thousands of Lines of Code into Simple Queries

by Dimitri Fontaine
Ask Me Two Questions!

Dimitri Fontaine
Citus Data
The Art of PostgreSQL

Turn Thousands of Lines of Code into Simple Queries