Designing your SaaS Database for Scale with Postgres

Lukas Fittl

Ozgun Erdogan

citusdata

What is Citus?

- Citus extends PostgreSQL (not a fork) to provide it with distributed functionality.
- Citus scales-out Postgres across servers using sharding and replication. Its query engine parallelizes SQL queries across many servers.
- Citus is open source: https://github.com/citusdata/citus

citusdata

When is Citus a good fit?

Three common use-cases:

1. Multi-tenant database: Citus allows you to scale out your multi-tenant (B2B) database to 100K+ tenants.
2. Real-time analytics: Citus enables ingesting large volumes of data and running analytical queries on that in human real-time.
3. NoSQL++: If you have high ingest requirements of 500K/sec, Citus can help you combine the power of structured and semi-structured data.

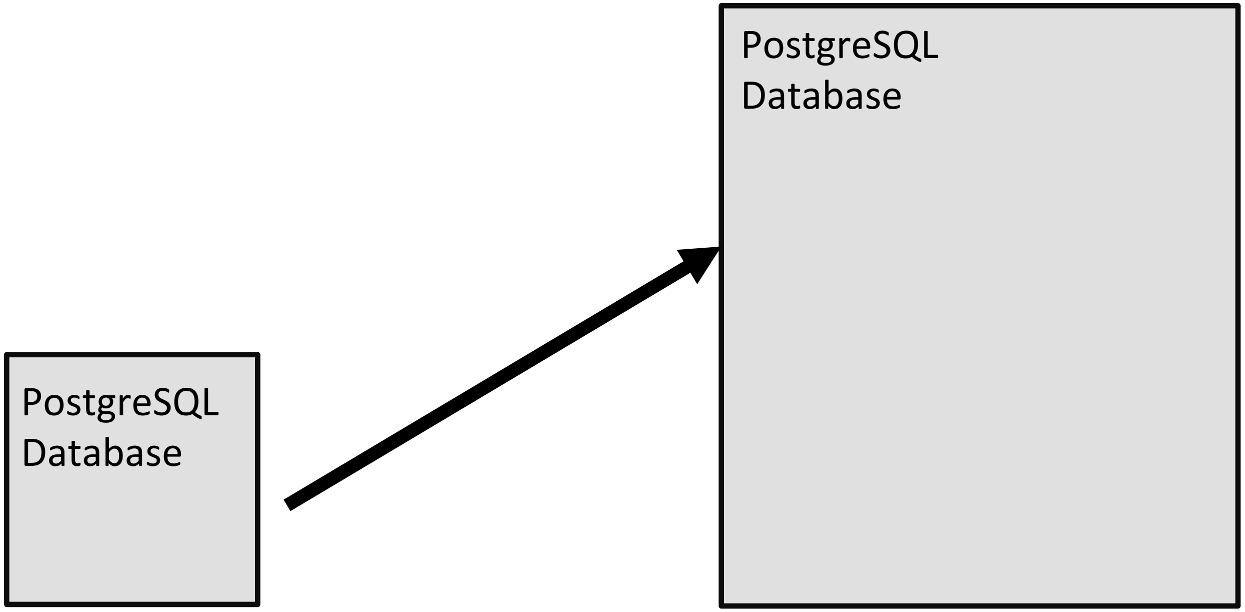
Talk Outline

1. Scaling Databases
2. Multi-tenant (SaaS) databases
3. Data Modeling
4. How to Scale Multi-tenant databases
5. Application Integration
6. Demo
7. Q & A

What does it mean to “scale”?

- Scaling: Allocating more resources to your application or database to improve performance.
- Scaling databases is harder than scaling apps.
- Types of resources you can scale:
    1. Software resources: Connections, number of processes
    2. Hardware resources: CPU, memory, and storage

Scaling Up Hardware



r3.xlarge

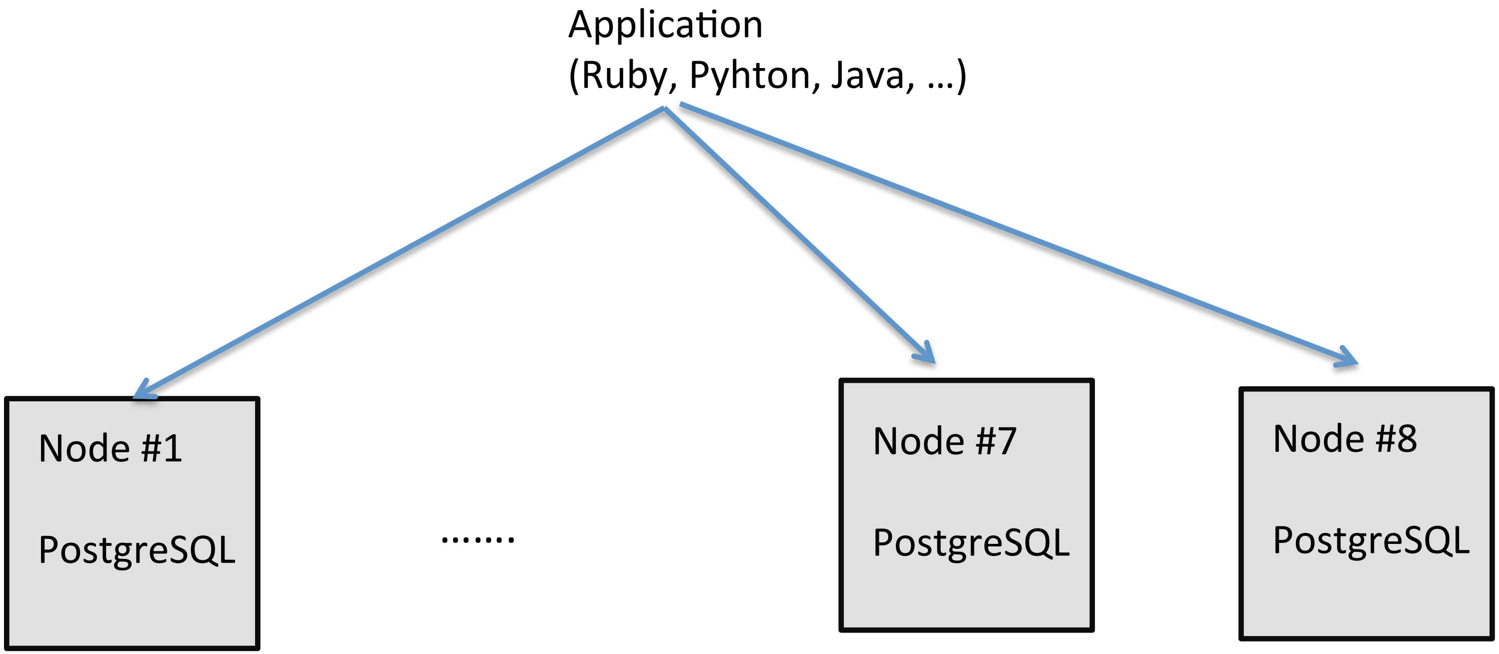
4 vCPUs
30 GB RAM
80 GB SSD

r3.4xlarge

16 vCPUs
120 GB RAM
320 GB SSD

citusdata

Scaling Out Hardware



Application
(Ruby, Pyhton, Java, …)

Node #1

PostgreSQL

…….

Node #7

PostgreSQL

Node #8

PostgreSQL

When is the right time to scale out

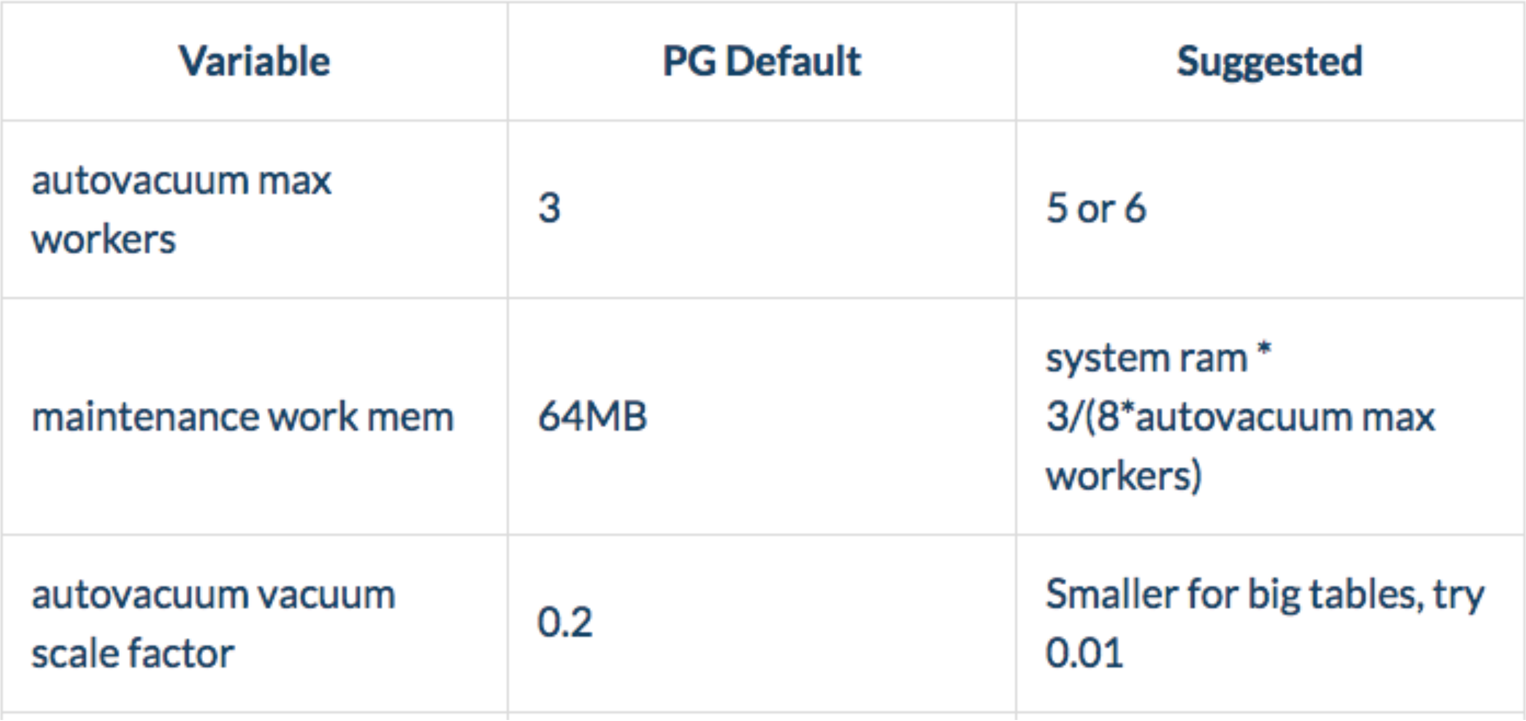
- Scaling up is easier than scaling out. If you can throw more hardware at the problem, that's the easiest way to scale.
- Also tune your database: http://pgconfsv.com/postgresql-when-its-not-your-day-job
- When is the right time to start thinking about scaling out?

Heuristic #1 on when to scale

- Your SaaS business is growing, you're on the second largest instance type available on your cloud / infrastructure provider
- Example tipping points
  - We signed a big customer, and now all our customers are hurting
  - One-off operational queries are bringing the database to a halt
  - We expect to grow by 10x next year

Heuristic #2

- Even after tuning, PostgreSQL’s autovacuum daemon can’t catch up with our write traffic



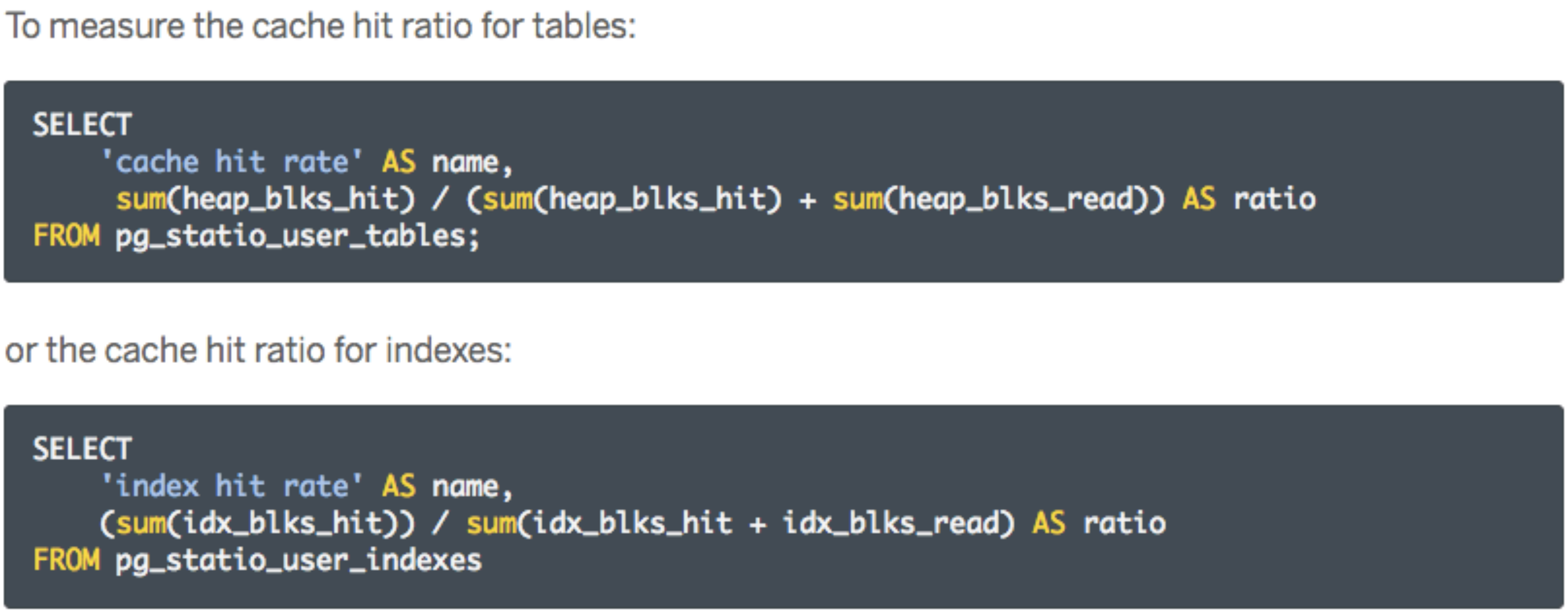
Variable	PG Default	Suggested
autovacuum max workers	3	5 or 6
maintenance work mem	64MB	system ram * 3/(8*autovacuum max workers)
autovacuum vacuum scale factor	0.2	Smaller for big tables, try 0.01

Heuristic #3

- Databases will cache recent and frequently accessed data in memory for you
- The database will track how often you use the cache and hit disk

- For OLTP applications, most of your working set should be fulfilled from the cache
  - Look to serve 99% from the cache

Heuristic #3 – cache hit ratio query



To measure the cache hit ratio for tables:

```
SELECT
    'cache hit rate' AS name,
     sum(heap_blks_hit) / (sum(heap_blks_hit) + sum(heap_blks_read)) AS ratio
FROM pg_statio_user_tables;
```

or the cache hit ratio for indexes:

```
SELECT
    'index hit rate' AS name,
    (sum(idx_blks_hit)) / sum(idx_blks_hit + idx_blks_read) AS ratio
FROM pg_statio_user_indexes
```

Source: Heroku -- Determining Cache Size

Plan ahead

- Plan ahead, optimize queries, and don’t wait until there isn’t another option

- When it’s time to scale out, you need to better understand your workload.
    1. B2B (multi-tenant databases) or B2C applications
    2. Transactional (OLTP) or analytical (OLAP)

What is a multi-tenant database

- If you're building a B2B application, you already have the notion of tenancy built into your data model
- B2B applications that serve other tenants / accounts / organizations use multi-tenant dbs
  - Physical service providers. For example, food services to other businesses
  - Digital service providers: Advertising, marketing, and sales automation

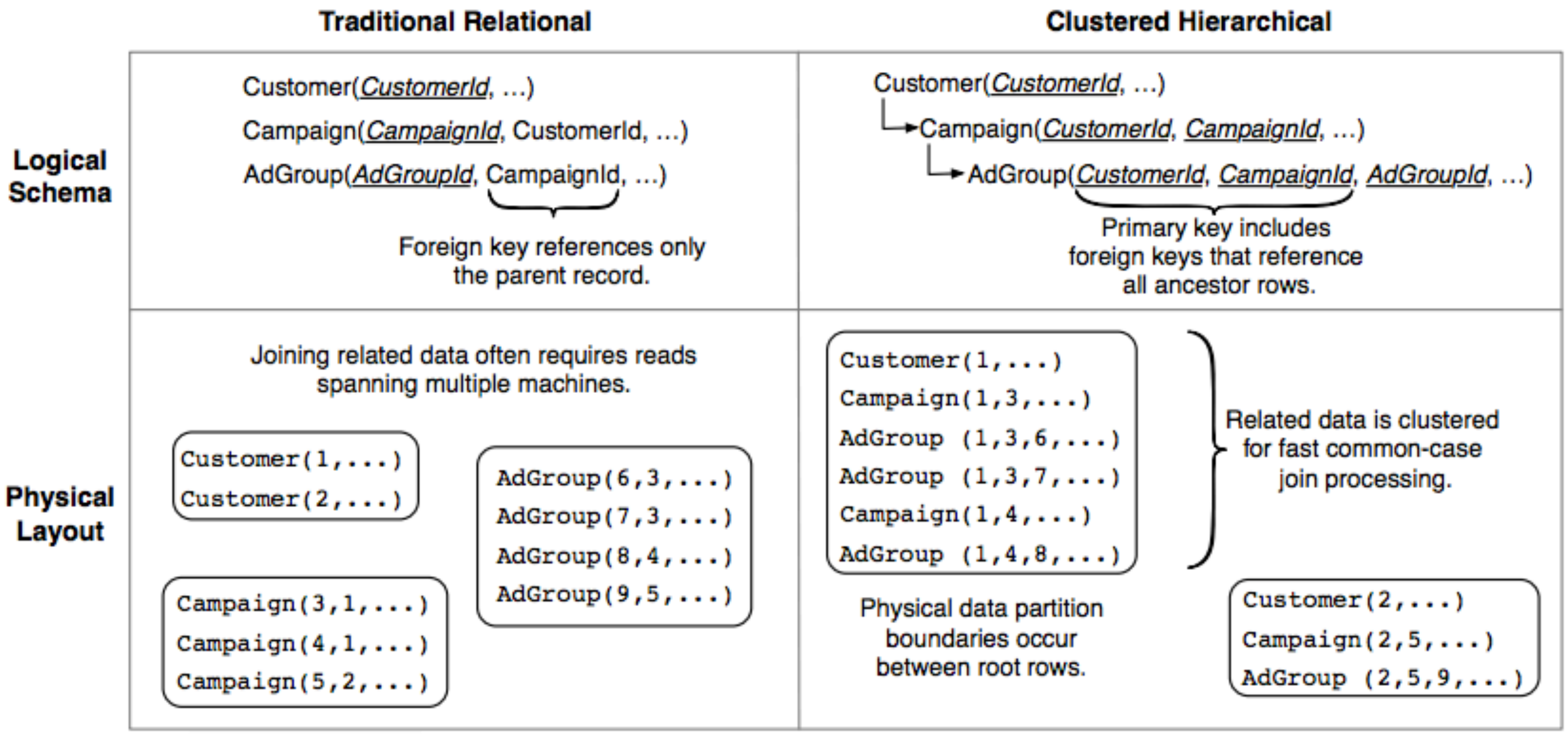
Trends in scaling multi-tenant apps

- Multi-tenant databases were commonplace in on-premises
- SaaS applications introduced the motivation to scale further
  - Cloud enables serving many smaller tenants
  - Instead of dozens of tenants, new SaaS apps reach to and handle 1K-100K tenants
  - Storage is cheap: You can store events or track a field's history

Google F1 – An Example

- Google F1 is an example that demonstrates a multi-tenant database.
- AdWords serves more than 1M tenants.
- F1 leverages key relational database features:
  - Transactions
  - Joins – avoid data duplication
  - Primary and foreign key constraints

Data modeling for multi-tenant



	Traditional Relational	Clustered Hierarchical

Logical Schema	Customer(*CustomerId*, ...)
	Campaign(*CampaignId*, CustomerId, ...)
	AdGroup(*AdGroupId*, CampaignId, ...)
	Foreign key references only the parent record.
		Customer(*CustomerId*, ...)
		→Campaign(*CustomerId*, *CampaignId*, ...)
		→AdGroup(*CustomerId*, *CampaignId*, *AdGroupId*, ...)
		Primary key includes foreign keys that reference all ancestor rows.

Physical Layout	Joining related data often requires reads spanning multiple machines.
	`Customer(1,...)`
	`Customer(2,...)`
	`AdGroup(6,3,...)`
	`AdGroup(7,3,...)`
	`AdGroup(8,4,...)`
	`AdGroup(9,5,...)`
	`Campaign(3,1,...)`
	`Campaign(4,1,...)`
	`Campaign(5,2,...)`
		`Customer(1,...)`
		`Campaign(1,3,...)`
		`AdGroup (1,3,6,...)`
		`AdGroup (1,3,7,...)`
		`Campaign(1,4,...)`
		`AdGroup (1,4,8,...)`
		Related data is clustered for fast common-case join processing.
		Physical data partition boundaries occur between root rows.
		`Customer(2,...)`
		`Campaign(2,5,...)`
		`AdGroup (2,5,9,...)`

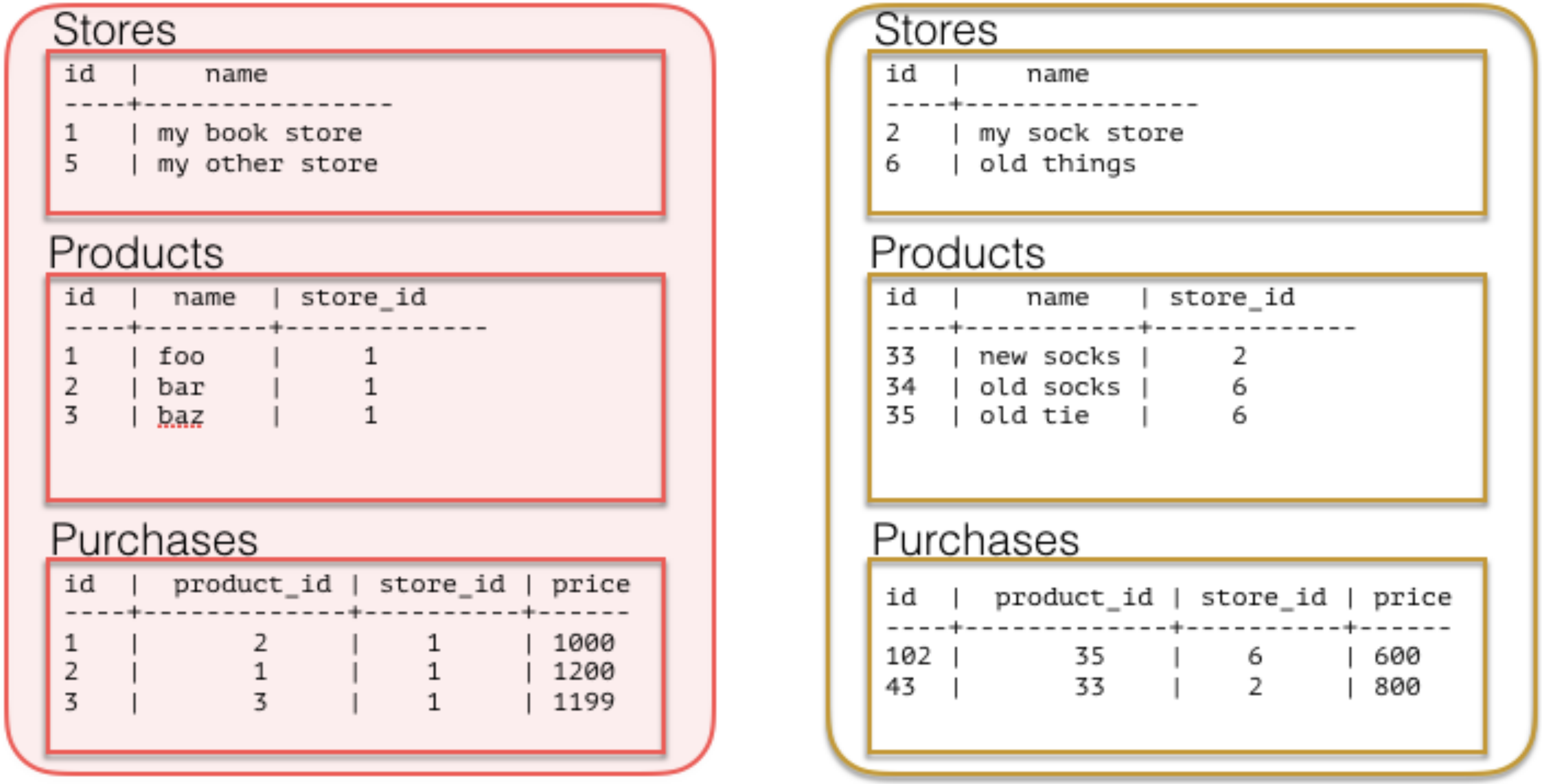
Figure 2: The logical and physical properties of data storage in a traditional normalized relational schema compared with a clustered hierarchical schema used in an F1 database.

Key Insight

- If you shard your tables on their primary key (in the relational model), then distributed transactions, joins, and foreign key constraints become expensive.

- Model your tables using the hierarchical database model by adding tenant_id. This colocates data for the same tenant together and dramatically reduces cost.

Concept of co-location



Stores

```
id  |     name
----+-----------------
1   | my book store
5   | my other store
```

Products

```
id  |  name    | store_id
----+----------+-------------
1   | foo      |       1
2   | bar      |       1
3   | baz      |       1
```

Purchases

```
id  |  product_id | store_id | price
----+-------------+----------+------
1   |         2   |     1    | 1000
2   |         1   |     1    | 1200
3   |         3   |     1    | 1199
```

Stores

```
id  |     name
----+---------------
2   | my sock store
6   | old things
```

Products

```
id  |     name    | store_id
----+-------------+-------------
33  | new socks |      2
34  | old socks |      6
35  | old tie     |      6
```

Purchases

```
id  |  product_id | store_id | price
----+-------------+----------+------
102 |        35   |    6     | 600
43  |        33   |    2     | 800
```

Does everything fit into hierarchical?

- What happens **if** I have a table that doesn't fit into the hierarchical database model?

1. Large table outside the hierarchy: Orgs and users that are shared across orgs
   - Shard on different column and don't join
2. Small table that is common to hierarchy
   - Create reference table replicated across all nodes

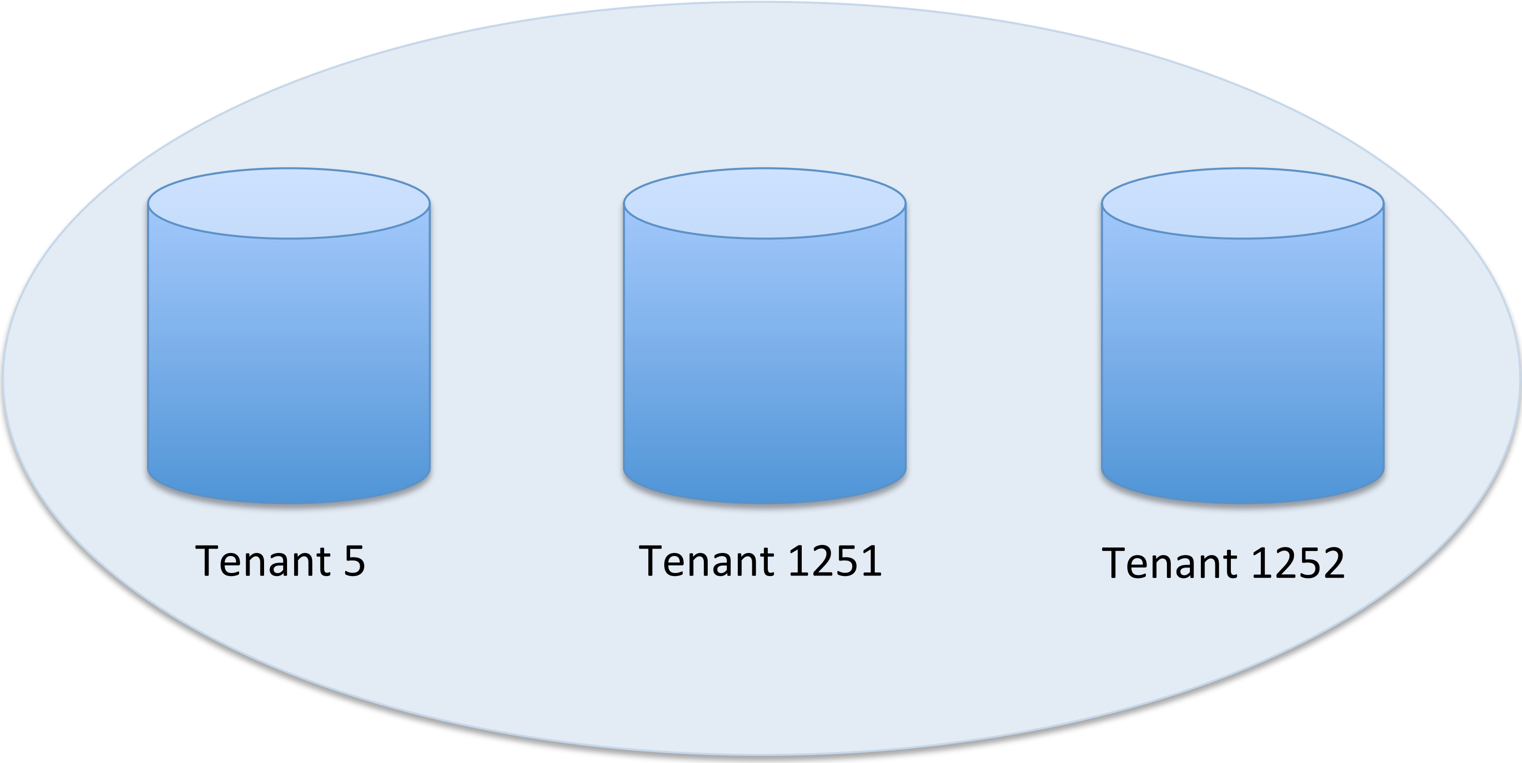
Scaling Multi-tenant Databases

- How to do you scale your multi-tenant database?
- Three high level options:

1. Create one database per tenant
2. Create one schema per tenant
3. Have all tenants share the same tables (and partition / shard tables)

citusdata

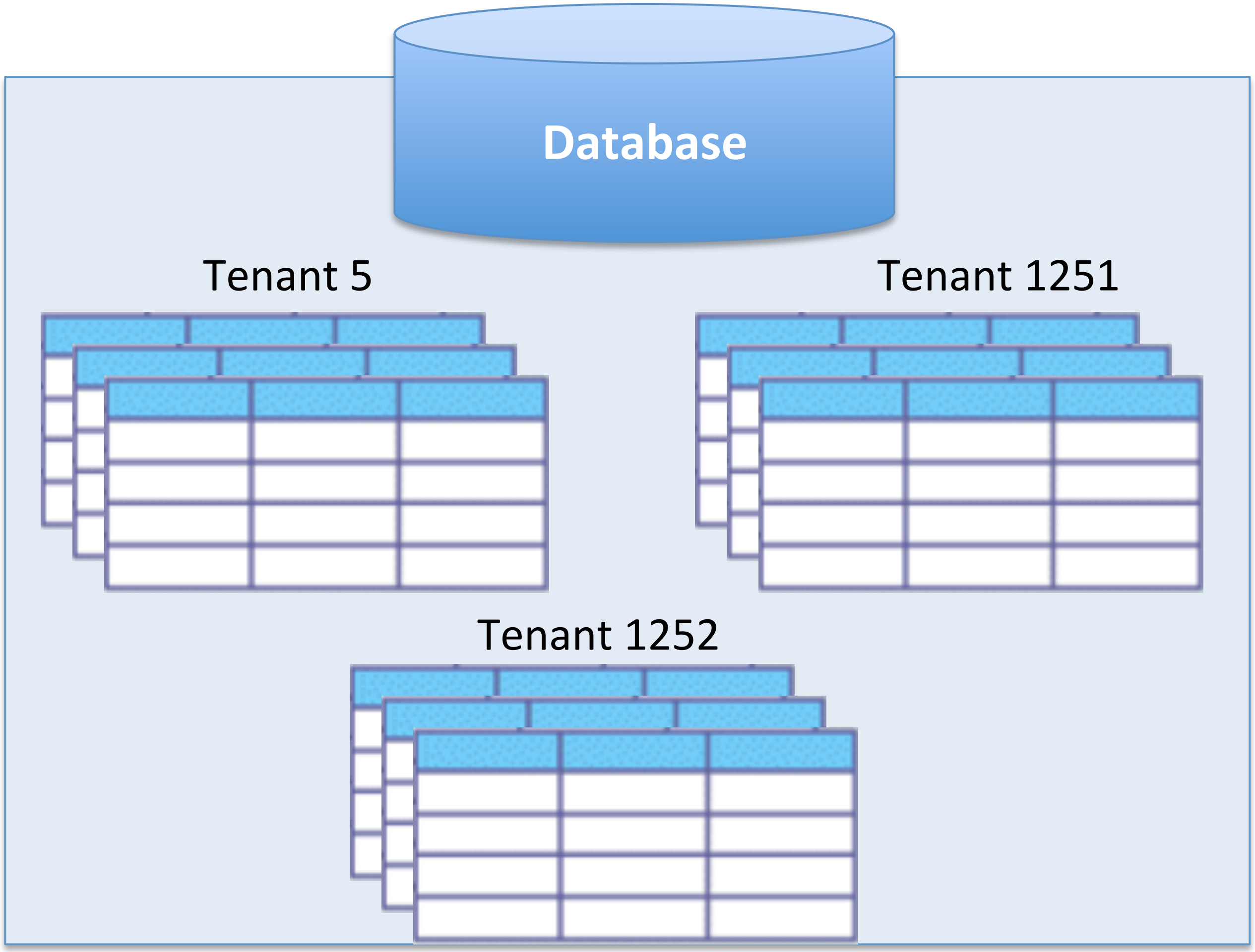
Create one database per tenant



Create one database per tenant

- Create a separate database for each tenant
- Isolation of tenants and more predictable compliance story
- DBA responsible for managing separate databases and resource allocation between them

Create one schema per tenant

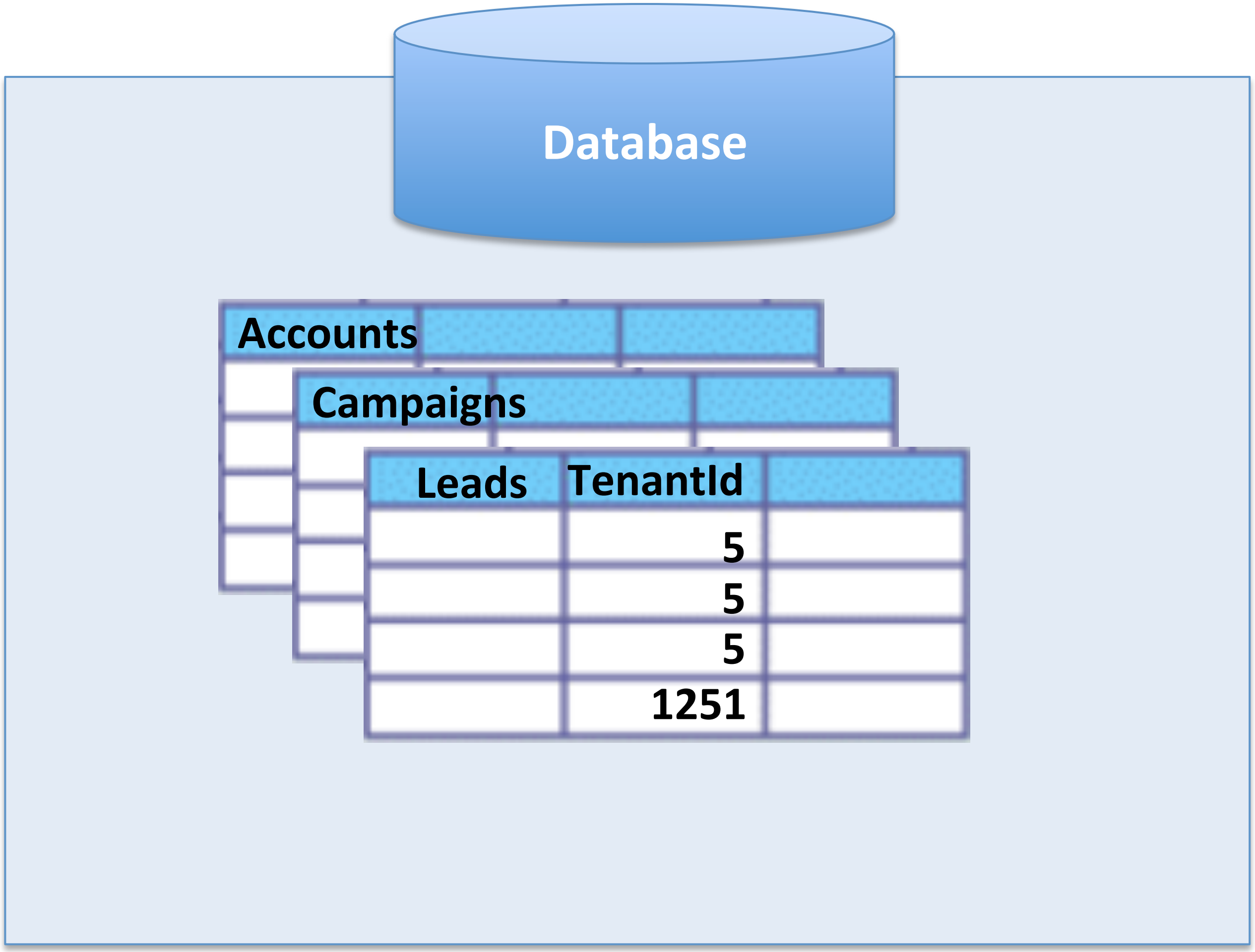


Create one schema per tenant

- Create a separate namespace (schema) for each tenant

- Isolate data / queries for one tenant in a schema. Make better use of resources than the “one database per tenant” model

Have all tenants share the same tables



Leads	TenantId	
	5	
	5	
	5	
	1251	

Have all tenants share the same tables

- Have all tenants share the same tables by adding a tenant_id column (and shard)

- Requires the application to control access to database, or row based access controls
- Scales to 1K-100K tenants through better resource sharing and simplifies operations and maintenance

Rule of thumb (simplified)

- Each design option can address questions around scale and isolation with enough effort. What's the primary criteria for you app?

- If you're building for scale: Have all tenants share the same table(s)
- If you're building for isolation: Create one database per tenant

Scaling: Resources

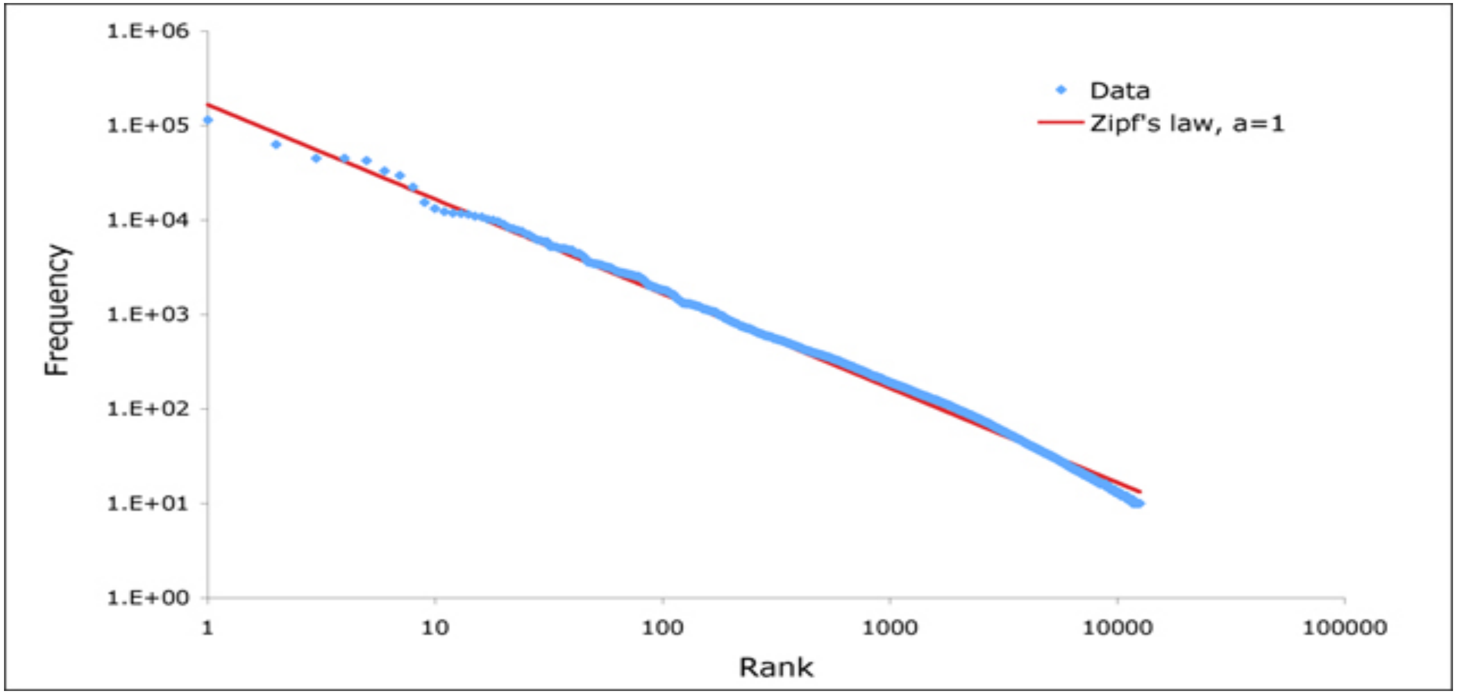
- If you create a separate database / schema for each tenant, you need to allocate resources to that database.
- Hardware: disk, memory, cpu, and network management
- Database software: shared_buffers, connection counts, backend processes
- ORM software: Cached information about databases / schemas

Scaling: Operational Simplicity

- Your database grows with your SaaS application.
- Schema changes (Alter Table … Add Column) and index creations (Create Index) are common operations.
- What happens when you have 10K tenants and you changed the schema for 5,000 of those tenants and observed a failure?

FAQ: How does largest tenant impact scale?

- Multi-tenant databases usually follow a Zipf / Power law distribution



FAQ: How does largest tenant impact scale?

- What percentage of the total data size belongs to the largest tenant?
- Guidelines around a standard Zipf distribution and different tenant counts:
  - 10 tenants: Largest tenant holds 60% of data (*)
  - 10K tenants: Largest tenant holds 2% of data (*)
- Look at your data's distribution to make informed scaling decisions

Application Integration

- Multi-tenant (B2B) Database Demo

Summary

- You can vertically or horizontally scale your database. Several heuristics help in deciding the right time to horizontally scale.
- To scale out a multi-tenant (B2B) database, picking the right distribution column and table colocation are key.
- There are three design patterns to scaling out a multi-tenant database. The “shared tables” approach offers the best scaling characteristics.

Q & A

Questions

www.citusdata.com/get_started

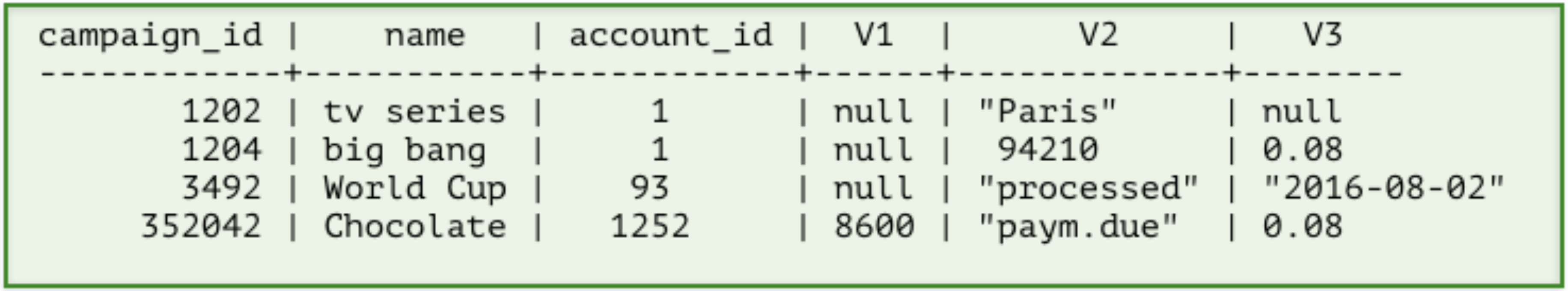
Appendix

FAQ: Data that varies across tenants

- What about data that varies across tenants?
- Different tenants / organizations may have their own needs that a rigid data model won't be able to address.
- One organization may need to track their stores in the US through their zip codes. Another customer in Europe may only want to keep tax ratios for each store.

FAQ: Salesforce Architecture

- If your tenants share the same table(s), one approach is creating a huge table with many string columns (Value0, Value1, …, Value500).

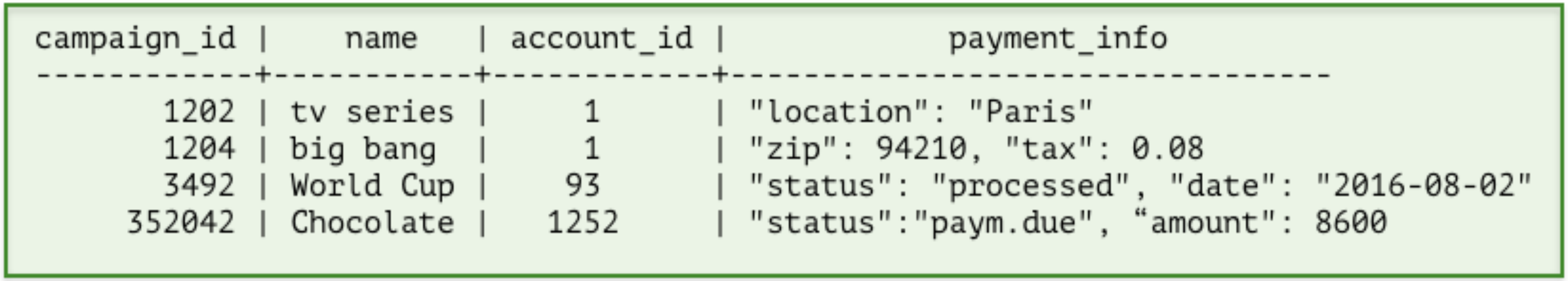


```
campaign_id |    name    | account_id |  V1  |      V2      |    V3
------------+------------+------------+------+--------------+---------
       1202 | tv series  |      1     | null | "Paris"      | null
       1204 | big bang   |      1     | null |  94210       | 0.08
       3492 | World Cup  |     93     | null | "processed"  | "2016-08-02"
     352042 | Chocolate  |    1252    | 8600 | "paym.due"   | 0.08
```

(*) Salesforce's multi-tenant arch: www.developerforce.com/media/ForcedotcomBookLibrary/Force.com_Multitenancy_WP_101508.pdf

FAQ: Semi-structured data types

- PostgreSQL has powerful semi-structured data types: hstore, json, and jsonb. These data types can express scalar, array, and nested fields.



```
campaign_id |    name    | account_id |              payment_info
------------+------------+------------+---------------------------------
       1202 | tv series |     1      | "location": "Paris"
       1204 | big bang  |     1      | "zip": 94210, "tax": 0.08
       3492 | World Cup |    93      | "status": "processed", "date": "2016-08-02"
     352042 | Chocolate |   1252     | "status":"paym.due", “amount": 8600
```