Cockroach Labs

# Build with CockroachDB

## The most highly evolved database on the planet

# Today’s Program

- Introduction to CockroachDB
- Architectural overview
- Install and run CockroachDB (hands-on)
- Demo: “Geo Tourist” app
  - Deploy on Kubernetes (K8s), using Operator
  - Resiliency in the face of node failure
  - Rolling upgrade -- zero downtime
- Serializable isolation and retry logic
- TypeORM
- Next steps

https://github.com/mgoddard/hackathon

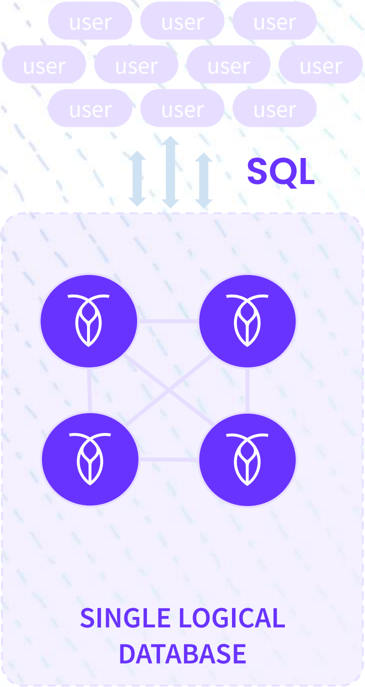
# CockroachDB

We re-architected the database from ground up to meet the demands of today's data-driven world.

| | | |
|---|---|---|
| Scale Fast | Survive Anything | Thrive Everywhere |

# CockroachDB: Scale Fast

Elastic & efficient scale for applications with a relational database



## Scale Fast

CockroachDB is a distributed, **relational database** that can be used for the most straightforward, common and high value workloads and gives your developers, **familiar standard SQL**

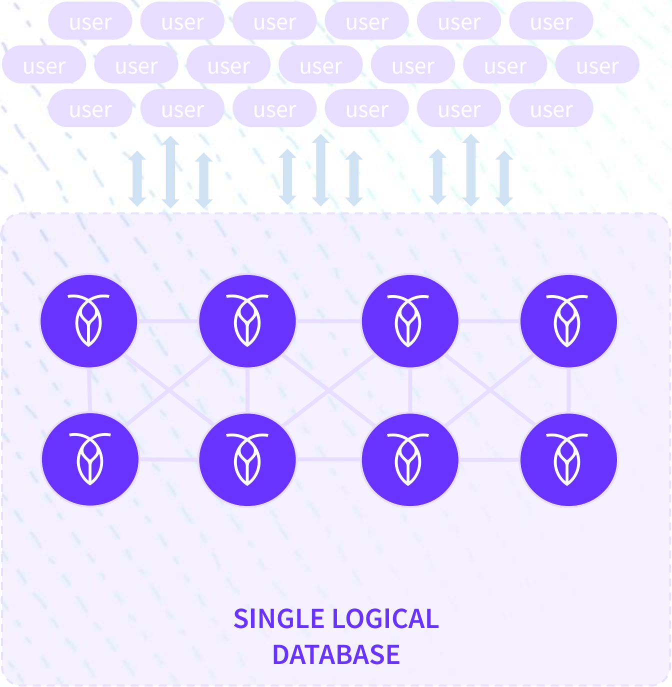
It is a database cluster that is comprised of nodes that appear as a single logical database

## Survive Anything

## Thrive Everywhere

# CockroachDB: Scale Fast

Elastic & efficient scale for applications with a relational database



## Scale Fast

**Scale** the database by simply adding more nodes

CockroachDB auto-balances to incorporate the new resource. No manual work is required.

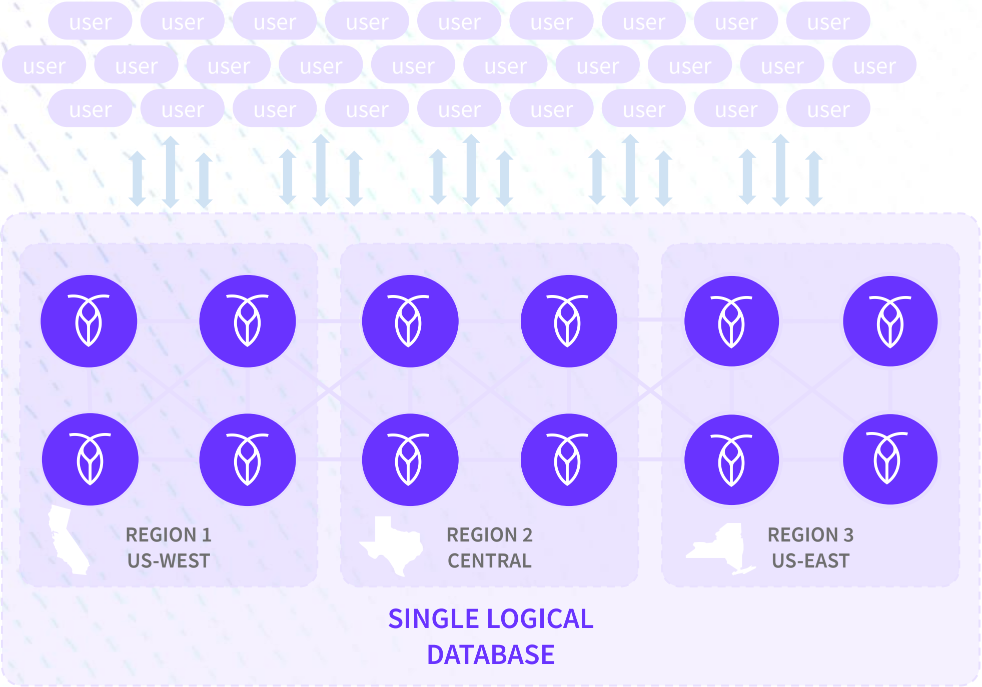
- Easy scale for increase in volume of data in the database
- Every node accepts reads & writes so you also scale transactional volume (writes)

## Survive Anything

## Thrive Everywhere

# CockroachDB: Scale Fast

Elastic & efficient scale for applications with a relational database



## Scale Fast

**Scale even further** across regions and even clouds, yet still deliver a single logical database

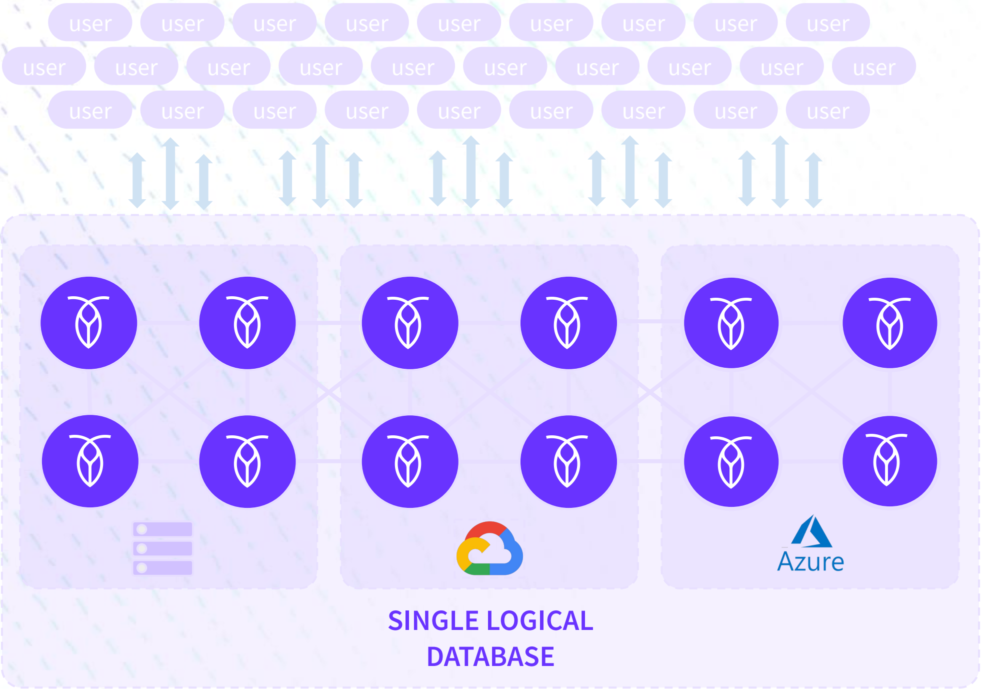
CockroachDB excels when deployed across multiple data centers in multiple regions

Survive Anything

Thrive Everywhere

# CockroachDB: Scale Fast

Elastic & efficient scale for applications with a relational database



## Scale Fast

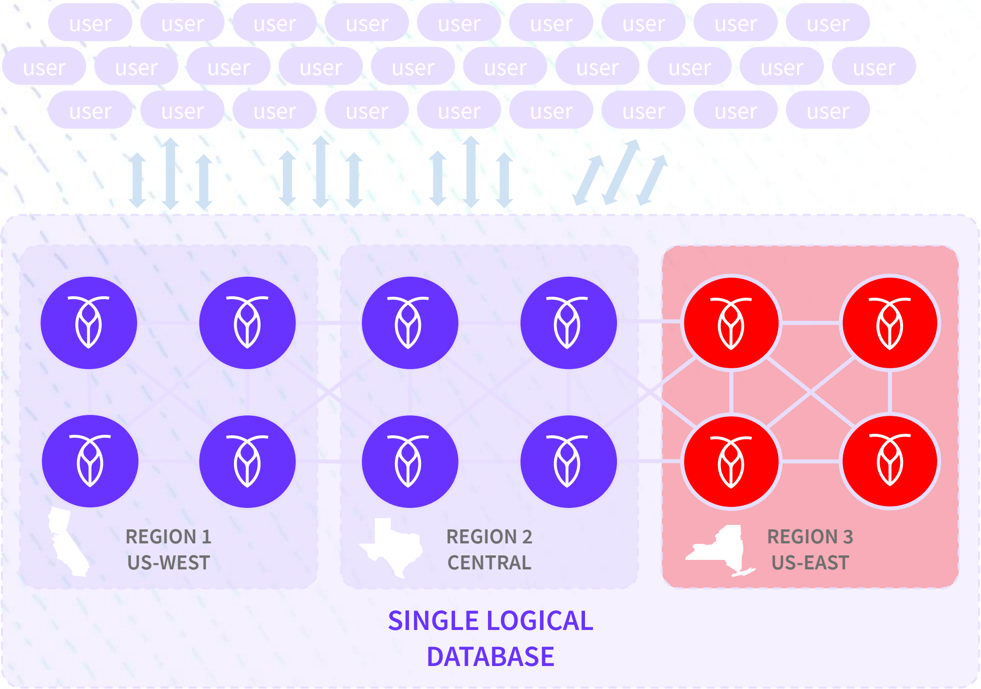
CockroachDB is the only database that allows you **span multiple public cloud providers** and on premise deployments with a single, logical database

Survive Anything

Thrive Everywhere

# CockroachDB: Survive Anything

A database that is always on & will survive any failure



Scale Fast

## Survive Anything

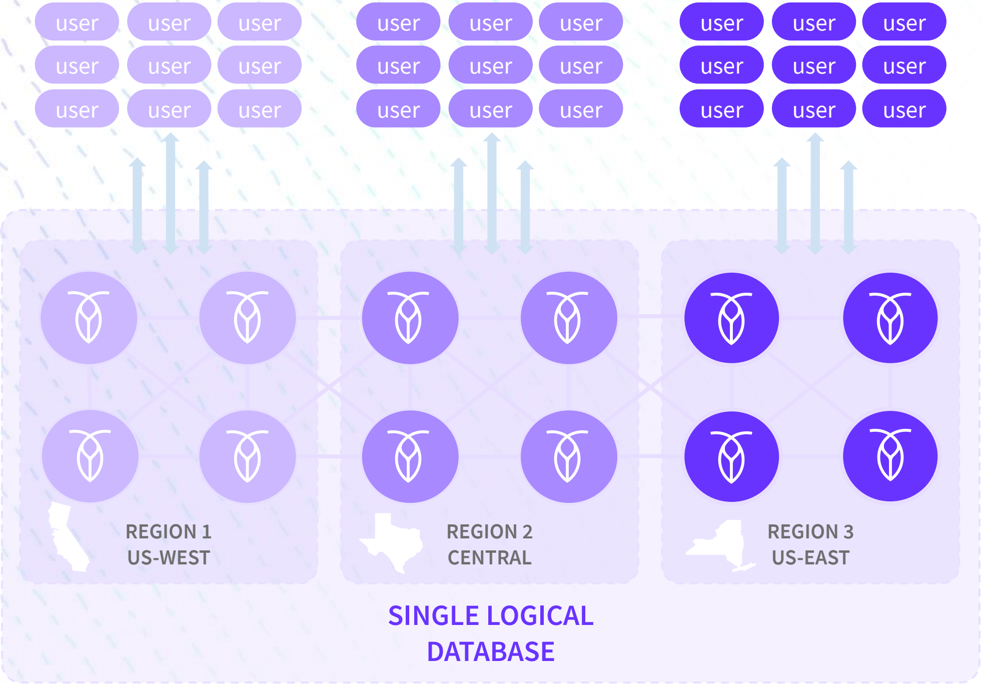
CockroachDB is naturally **resilient** so you can survive failure of a node or even an entire region without service disruption

- Always-on and with zero RPO
- Allows for no downtime rolling upgrades
- Online schema changes

Thrive Everywhere

# CockroachDB: Thrive Everywhere

Meet user needs even in the most broadly dispersed environment



Scale Fast

Survive Anything

## Thrive Everywhere

CockroachDB allows you to tie each row to a physical location based on data within each record

- reduce read/write latencies
- comply with regulations
- ensure customer data privacy

# CockroachDB works the way you work

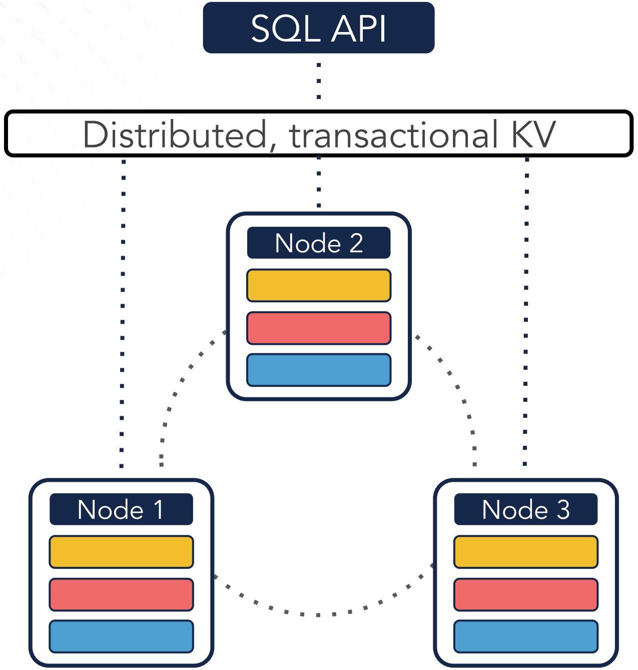
## Delivers enterprise features to ensure it fits into your environment and processes

| | | |
|---|---|---|
| Integration | Postgres wire-compatible, Change data capture | Key enterprise features expected in production environments |
| Management | DB Console, Rolling Upgrades, Distributed Backup/Restore, Prometheus | |
| Optimization | Cost Based Optimizer, Query Inspection, Online schema changes | |
| Security | RBAC, Kerberos, Encryption at rest and in motion (TLS) | |
| Deployment | On Prem, Multi-cloud, Hybrid cloud, Fully Managed | Ensure your success with architecture & deployment |
| Services | Architecture, Sizing, Deployment, Operational Excellence | |
| Training | Free Cockroach University, In person training | World class documentation and training gets you up to speed |
| Documentation | WORLD class, comprehensive documentation | |

# Architecture Overview

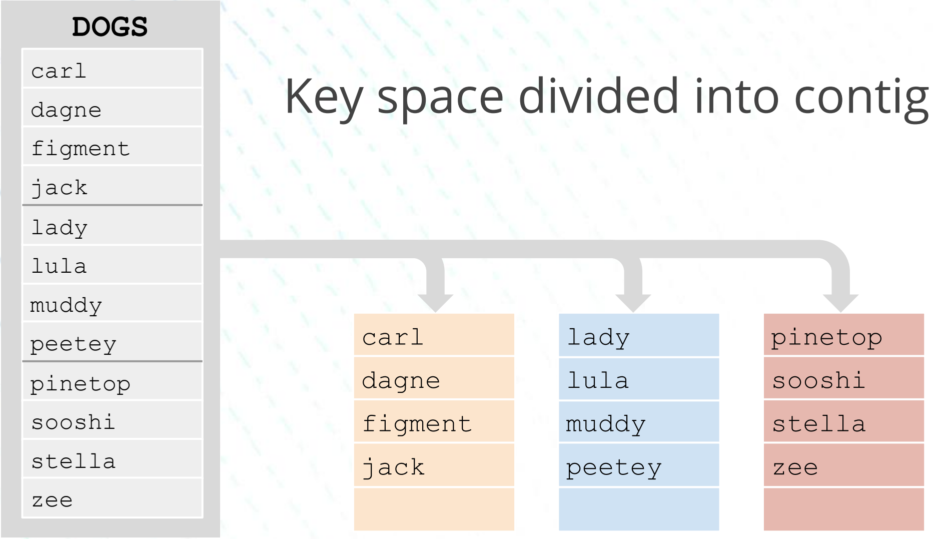
# Architecture Overview

- Dynamically Distributed Data
    - Elasticity
    - Resilient Self Healing
- Geo-Partitioning
    - Fast local latency
    - Globally available data
- PostgreSQL wire protocol
    - Postgres Drivers and ORMs
    - Tools : DBeaver, DataGrip, ...



# Monolithic Key Space -> Divided into RANGES

Key space divided into contiguous ~512MB ranges



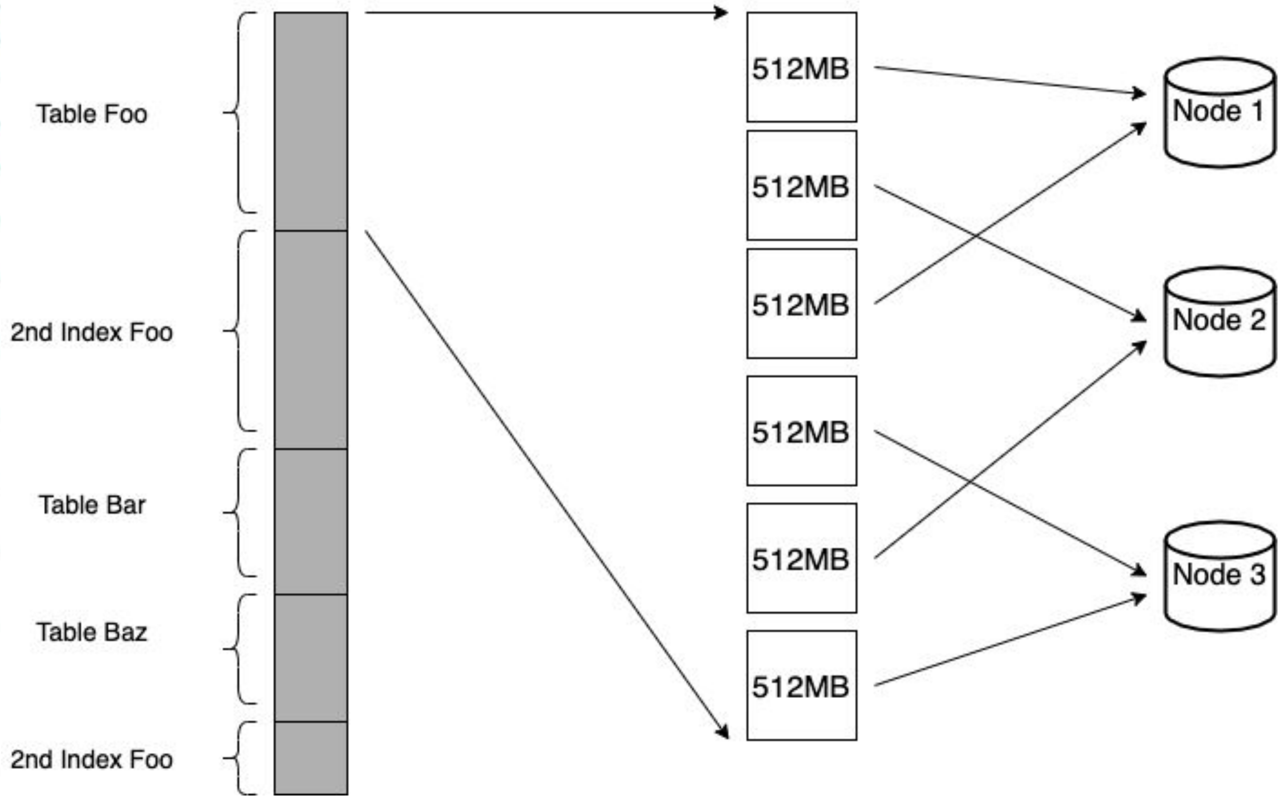
| DOGS |
|---|
| carl |
| dagne |
| figment |
| jack |
| lady |
| lula |
| muddy |
| peetey |
| pinetop |
| sooshi |
| stella |
| zee |

| | | |
|---|---|---|
| carl | lady | pinetop |
| dagne | lula | sooshi |
| figment | muddy | stella |
| jack | peetey | zee |
| | | |

Ranges are small enough to be moved/split quickly

Ranges are large enough to amortize indexing overhead

# RANGES MAP TO NODES



# What’s the magic in the database architecture?

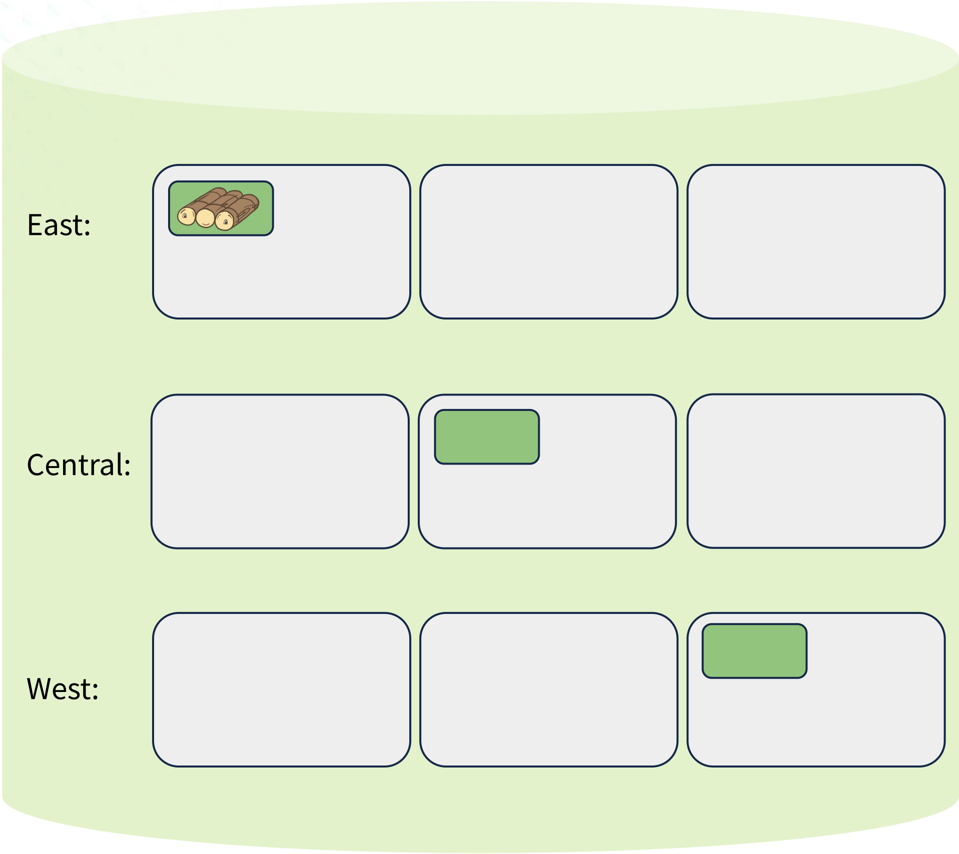
Every Node (server) Is A Gateway

Table data stored in 512MB Ranges...
...and sorted by primary key

Apple
Banana
Carrot

Ranges are replicated (3x default)

A Range set has an elected
Leaseholder & RAFT Leader



# How are replicas placed?

*Leaseholders will naturally move to where there is load*

Space

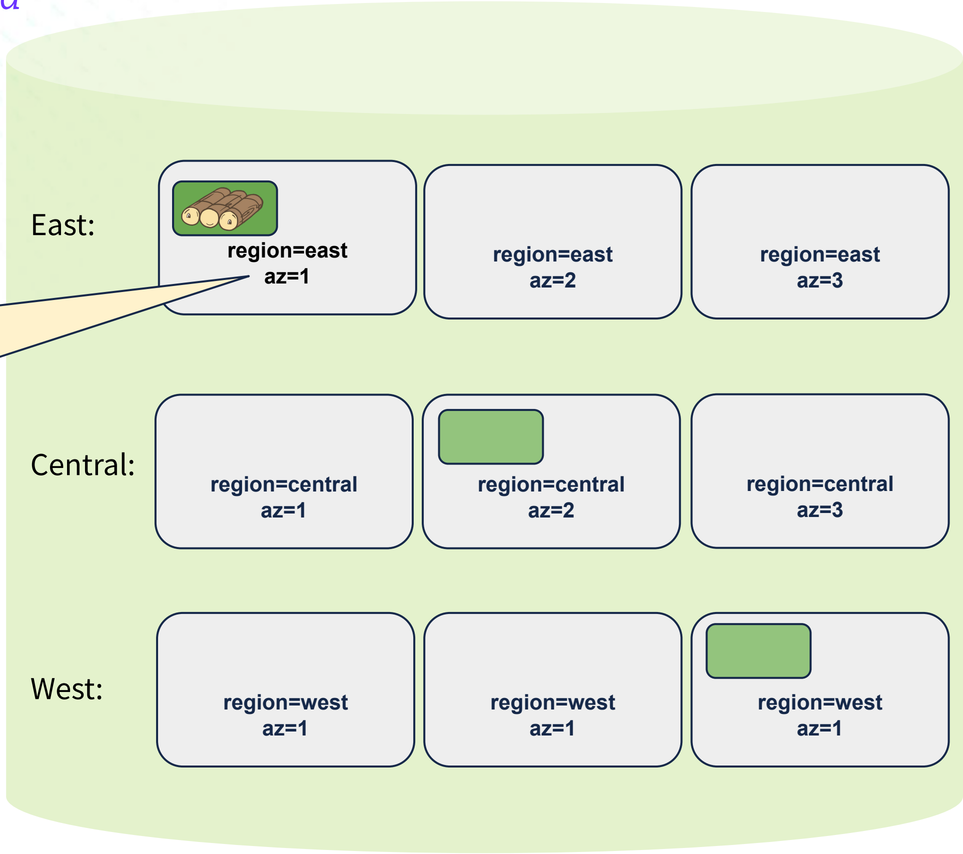
**Diversity**

Load

Latency

Deterministic

Node locality flags on your nodes will ensure data is evenly spread by default



East:
region=east az=1
region=east az=2
region=east az=3

Central:
region=central az=1
region=central az=2
region=central az=3

West:
region=west az=1
region=west az=1
region=west az=1

# How are replicas placed?

*Leaseholders will naturally move to distribute load*

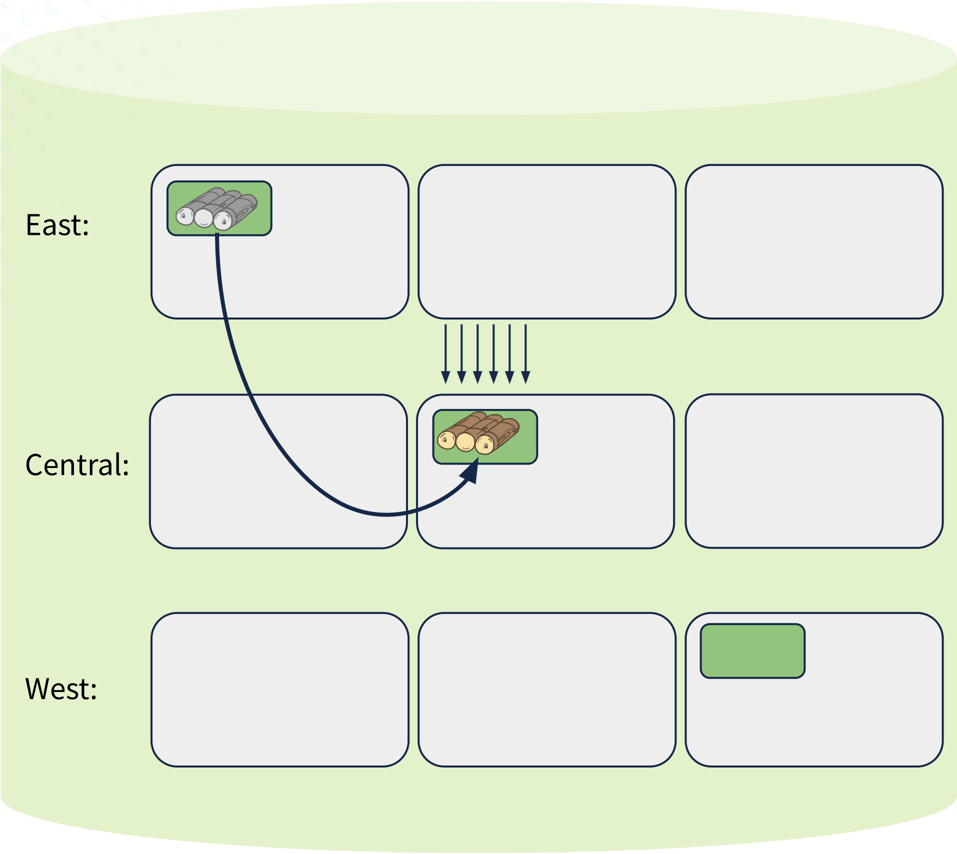
Space

Diversity

**Load**

Latency

Deterministic



# How are replicas placed?

*Leaseholders can be elected into other replicas to reduce query latency*

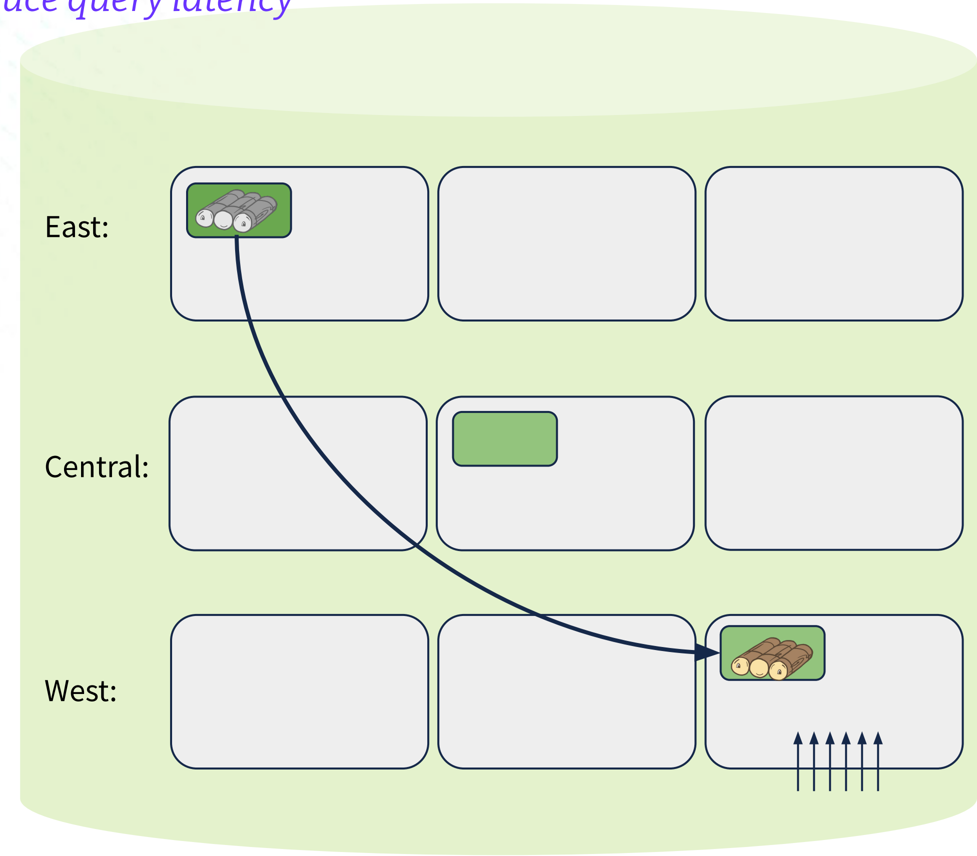
Space

Diversity

Load

**Latency**

Deterministic



# How are replicas placed?

*By default, Cockroach replicates data for you using the following heuristics*

Space

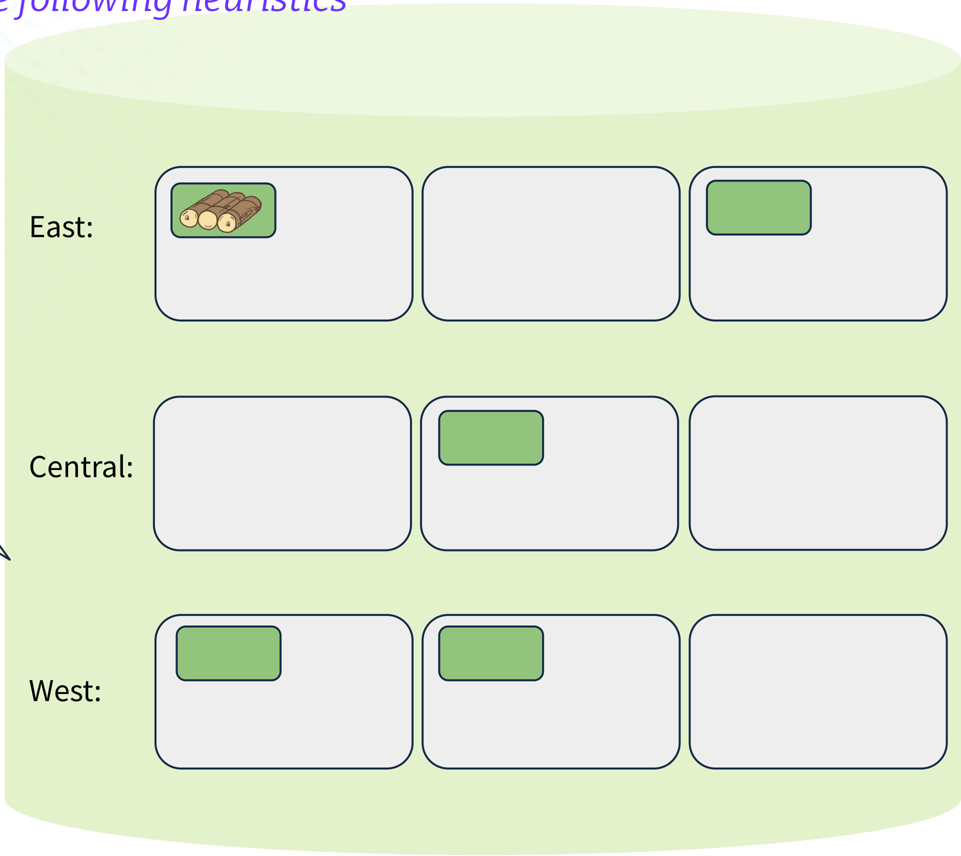
Diversity

Load

Latency

**Deterministic**

**ALTER RANGE myZone…**
num_replicas = 5
lease_preferences = ‘East’
constraints = +East:2, +Central:1, +West:2
(*What is the RF value  in this example?*)



# Install and run CockroachDB

# Install CockroachDB locally

Install Homebrew:

```
$ /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Use Homebrew to install CockroachDB:

```
$ brew install cockroachdb/tap/cockroach
```

For other installation options, see CockroachDB docs.

# Start a three-node cluster
## (on your own computer)

```
cockroach start --insecure --listen-addr=localhost:26257
--join=localhost:26257,localhost:26258,localhost:26259 \
  --http-addr=localhost:8080 --store=cockroach-data-1 --background

cockroach start --insecure --listen-addr=localhost:26258
--join=localhost:26257,localhost:26258,localhost:26259 \
  --http-addr=localhost:8081 --store=cockroach-data-2 --background

cockroach start --insecure --listen-addr=localhost:26259
--join=localhost:26257,localhost:26258,localhost:26259 \
   --http-addr=localhost:8082 --store=cockroach-data-3 --background

cockroach init --host localhost:26258 --insecure
```

View the Admin UI:

```
open http://localhost:8080
```

# Connect using the SQL shell

In a terminal window, connect using the SQL shell:

```
cockroach sql --insecure
```

To verify that you did this correctly, run a SHOW DATABASES command:

```
SHOW DATABASES;
```

The output should look something like this:

```
 database_name | owner | primary_region | regions | survival_goal
----------------+-------+----------------+---------+-----------------
  defaultdb     | root  | NULL           | {}      | NULL
  postgres      | root  | NULL           | {}      | NULL
  system        | node  | NULL           | {}      | NULL
```

# Create a table

In the SQL shell, create a table called `products` with the following columns:

- `id` of type `UUID` (make this the primary key)
- `name` of type `STRING`
- `quantity of type INT`
- `price` of type `DECIMAL`

To verify that you've done this correctly, run the `SHOW CREATE TABLE` command. You should see each of these columns, along with its type.

# Add a node to scale the cluster

Add the new node:

```
cockroach start --insecure --listen-addr=localhost:26260 \
  --join=localhost:26257,localhost:26258,localhost:26259 \
  --http-addr=localhost:8083 --store=cockroach-data-4 --background
```

Check the DB Console to see that it joined the cluster:

http://localhost:8080/#/overview/list

# **Fault Tolerance:** Node or AZ Failure

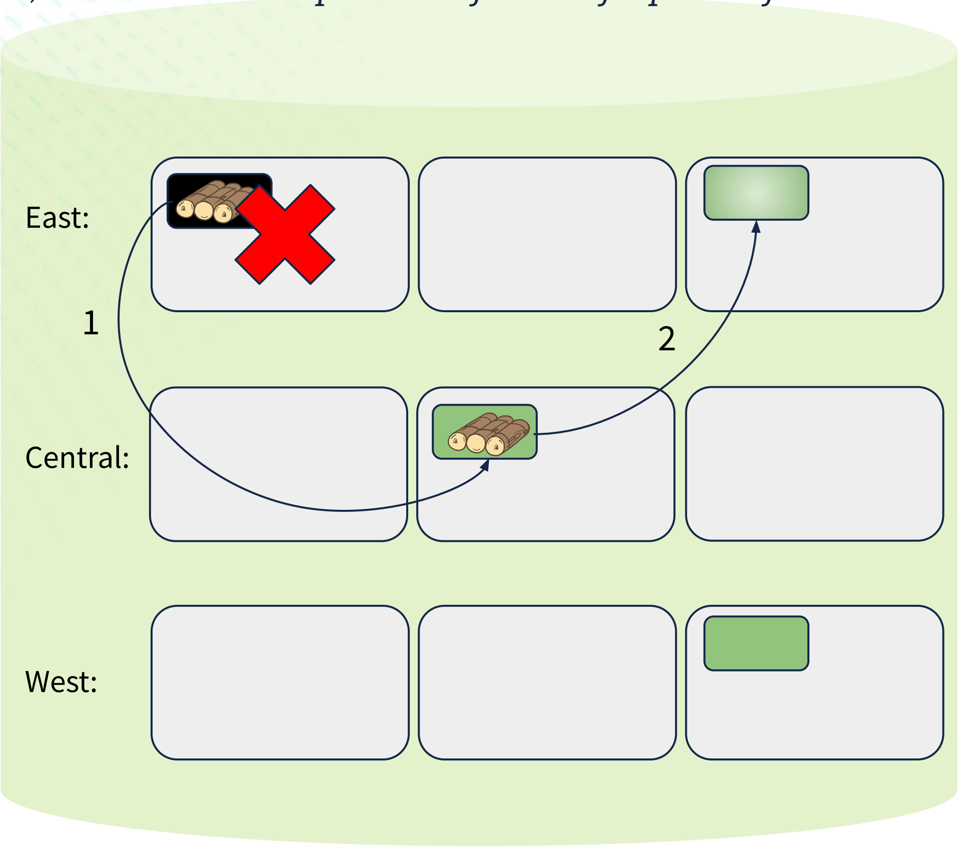
*CockroachDB can survive **(n - 1)/2** failures, where **n** is the replication factor of a piece of data.*

1) A new leaseholder gets elected after a period of 4 to 9 seconds.

   By default, the cluster waits for the node to come back within 5 minutes. The 5 minutes is configurable. If the node returns, the range will get caught up by either receiving new data or by comparing RAFT logs and catching itself up.

2) If the node hasn't rejoined within 5 minutes, it will be marked "Dead" and the up-replication process will begin.

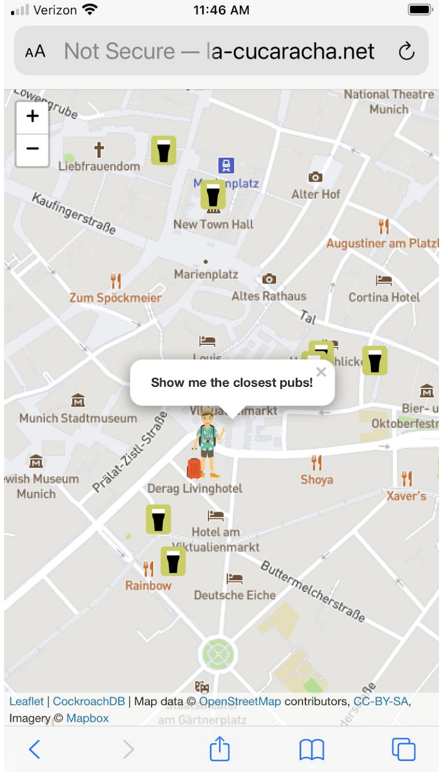
   If we have a "Return from the Dead" node, it will be added as a fresh new node and immediately start serving traffic and having data replicated to it.



# Demo: Geo Tourist app (GitHub repo)

## About 20 minutes

- Deploy using CockroachDB K8s Operator on a ***GKE*** cluster
- Add a DB user for the app (user: “tourist”, password: “tourist”)
- Deploy data loader pod, load 475k rows
- Log into DB Console
- Start the web app, get LB IP address, connect using browser
- ***Kill a node***, verify app continues to run, users unaffected
- Perform a ***rolling upgrade***, observe app running



Verizon 11:46 AM

AA Not Secure — la-cucaracha.net

Show me the closest pubs!

Leaflet | CockroachDB | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

The demo is based on this script.

# Lab: Serializable Isolation

# Serializable Isolation

- ***Write skew***: a transaction reads something, makes a decision based on the value it read, and writes to the database. However, by the time the write is made, the premise driving the decision is no longer true.
- Postgres: default isolation level of READ COMMITTED, which can result in write skew.
- SERIALIZABLE isolation guarantees that, even though transactions may execute in parallel, the result is the same as if they had executed one at a time, without any concurrency.
- CockroachDB provides (only) SERIALIZABLE isolation.
- One consequence of this is the need to implement transaction retry logic within application code. StackOverflow discussion of retry logic and TypeORM.

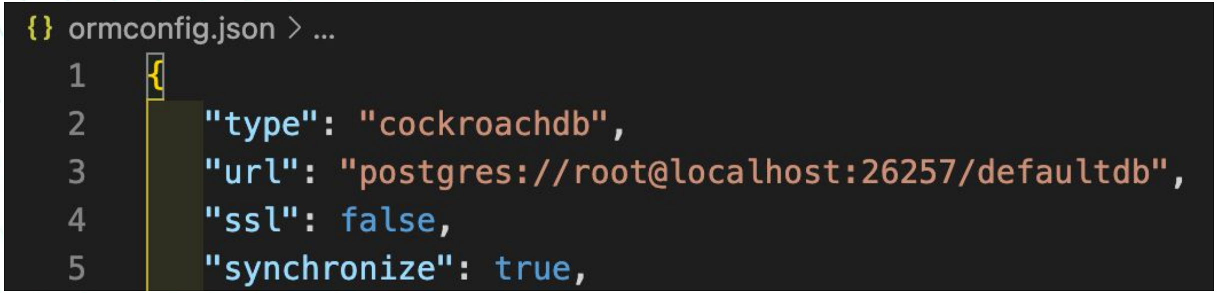
Hands on: https://www.cockroachlabs.com/docs/stable/demo-serializable.html

# Lab: Get set up with TypeORM

# Set up TypeORM for Node.js / TypeScript

We'll use the TypeORM intro for this: [https://typeorm.io/#/](https://typeorm.io/#/)

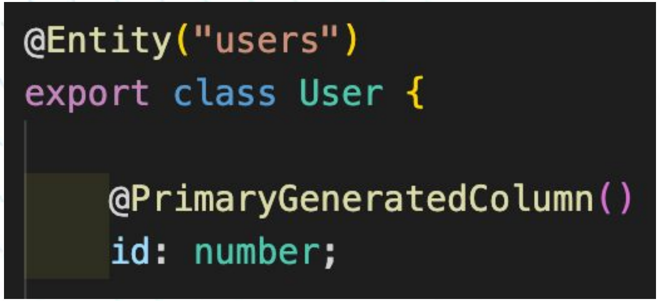
- Start with the # Installation section
- Next, run through # Quick Start
    - You can connect to your own local CockroachDB installation:



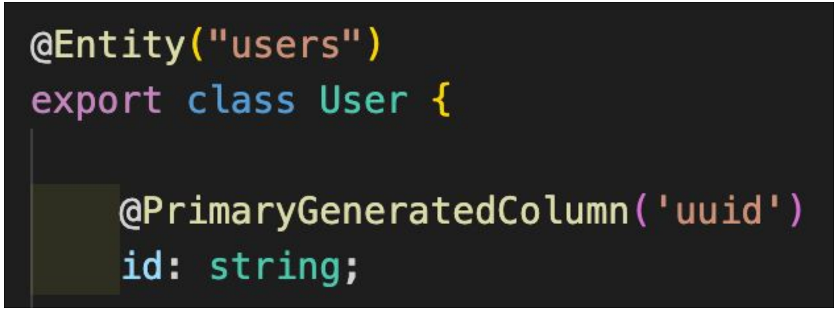
```
{
    "type": "cockroachdb",
    "url": "postgres://root@localhost:26257/defaultdb",
    "ssl": false,
    "synchronize": true,
```

# Avoid INT / Sequence for Primary Key

- Recall how CockroachDB organizes data: in *ranges*, where each range has a *leaseholder* residing on a node, and the data is ordered by primary key.
- Given that, what could happen if your app uses sequential primary keys?
- This is easily avoided: choose UUID as the primary key type.
- Try it in your User entity.  What is the resulting table definition?

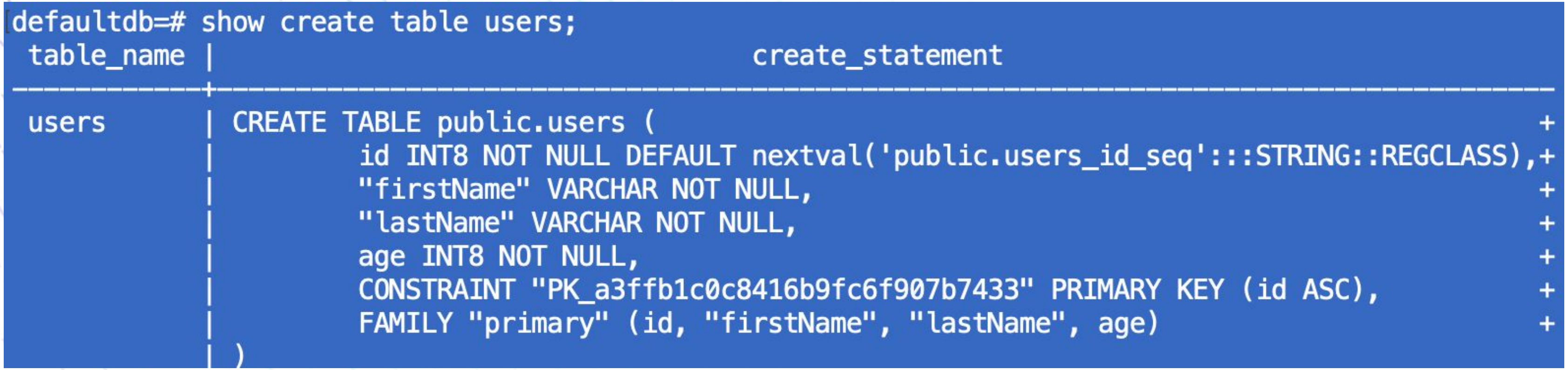


```
@Entity("users")
export class User {

    @PrimaryGeneratedColumn()
    id: number;
```

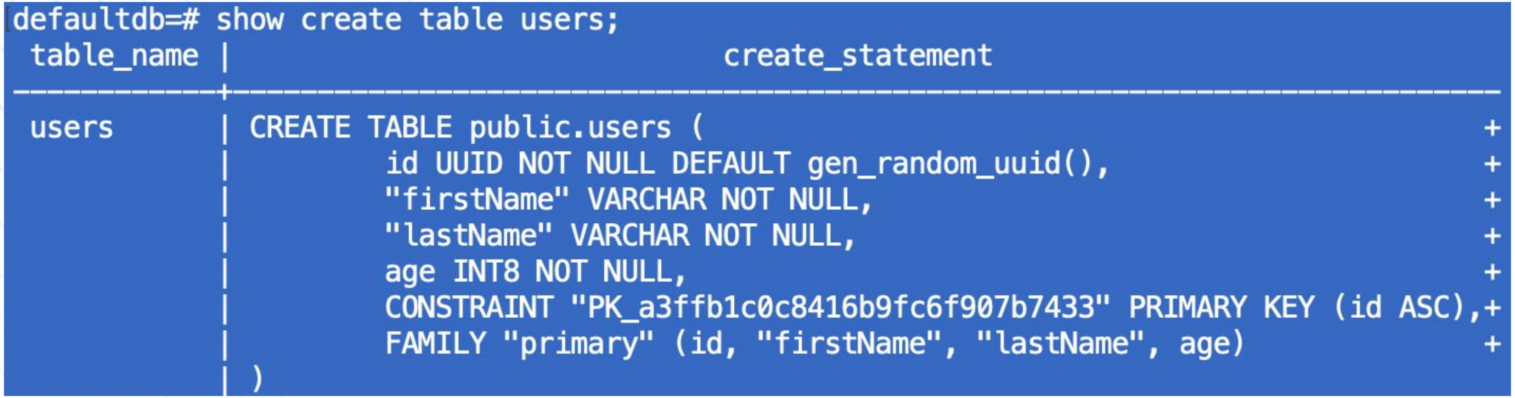


```
@Entity("users")
export class User {

    @PrimaryGeneratedColumn('uuid')
    id: string;
```

# TypeORM did what we expected!



```
defaultdb=# show create table users;
  table_name |                                   create_statement
-------------+-------------------------------------------------------------------------------------
  users      | CREATE TABLE public.users (                                                         +
             |         id INT8 NOT NULL DEFAULT nextval('public.users_id_seq':::STRING::REGCLASS),+
             |         "firstName" VARCHAR NOT NULL,                                               +
             |         "lastName" VARCHAR NOT NULL,                                                +
             |         age INT8 NOT NULL,                                                          +
             |         CONSTRAINT "PK_a3ffb1c0c8416b9fc6f907b7433" PRIMARY KEY (id ASC),           +
             |         FAMILY "primary" (id, "firstName", "lastName", age)                         +
             | )
```



```
defaultdb=# show create table users;
  table_name |                              create_statement
-------------+-----------------------------------------------------------------------------
  users      | CREATE TABLE public.users (                                                 +
             |         id UUID NOT NULL DEFAULT gen_random_uuid(),                         +
             |         "firstName" VARCHAR NOT NULL,                                       +
             |         "lastName" VARCHAR NOT NULL,                                        +
             |         age INT8 NOT NULL,                                                  +
             |         CONSTRAINT "PK_a3ffb1c0c8416b9fc6f907b7433" PRIMARY KEY (id ASC),+
             |         FAMILY "primary" (id, "firstName", "lastName", age)                 +
             | )
```

# What Next?
# You could ...

- Continue with the "Step-by-Step Guide" section of the TypeORM guide
- Review the AS OF SYSTEM TIME clause ("time travel" queries, aka "AOST")
- Add AOST to TypeORM (see https://github.com/typeorm/typeorm/issues/4646)
- Explore multi-region capabilities: https://github.com/chriscasano/multi-region-dad-jokes
- Explore any of several interesting topics available at this jumping off point: https://github.com/cockroachlabs/workshop_labs
- Make the "Geo Tourist" demo more interesting: https://github.com/cockroachlabs-field/crdb-geo-tourist

***Whatever you choose, we're confident you'll love building it on CockroachDB!***

# Questions?

# Resources

- CockroachDB Public Slack
- CockroachDB Docs
- Cockroach University