DataStax

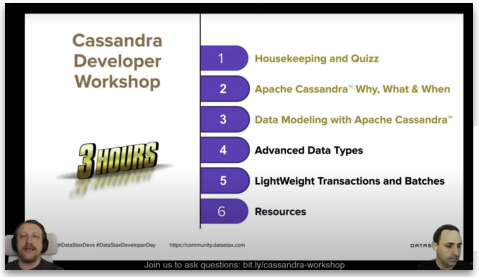
From SQL to NoSQL
A Migration Path

Using the petclinic app and Apache Cassandra on Astra

LEVEL UP with the DataStax Developers

Housekeeping

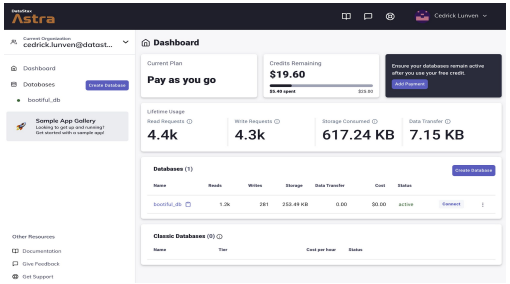
Livestream: youtube.com/DataStaxDevs



YouTube

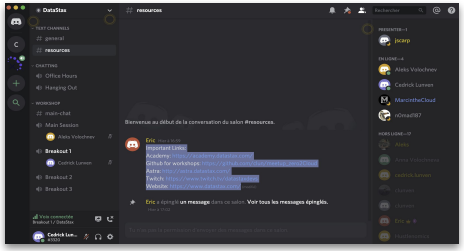
Twitch

Runtime: dtsx.io/workshop



DataStax Astra

Questions: bit.ly/cassandra-workshop

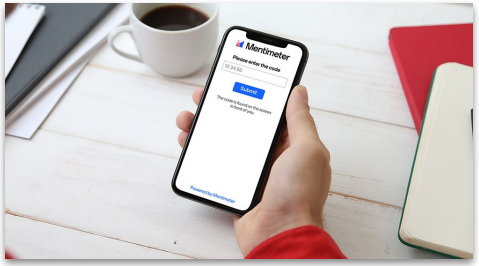


Discord

YouTube

Available on the iPhone App Store

Quizz: menti.com

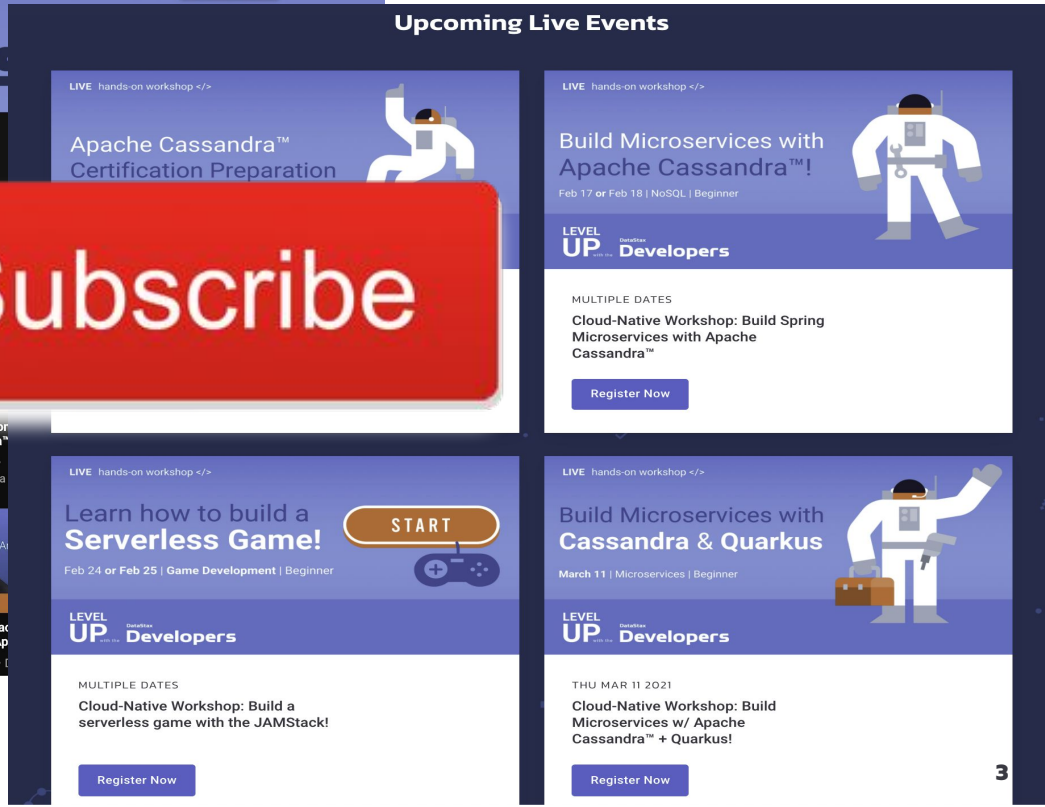


Mentimeter

GET IT ON Google play

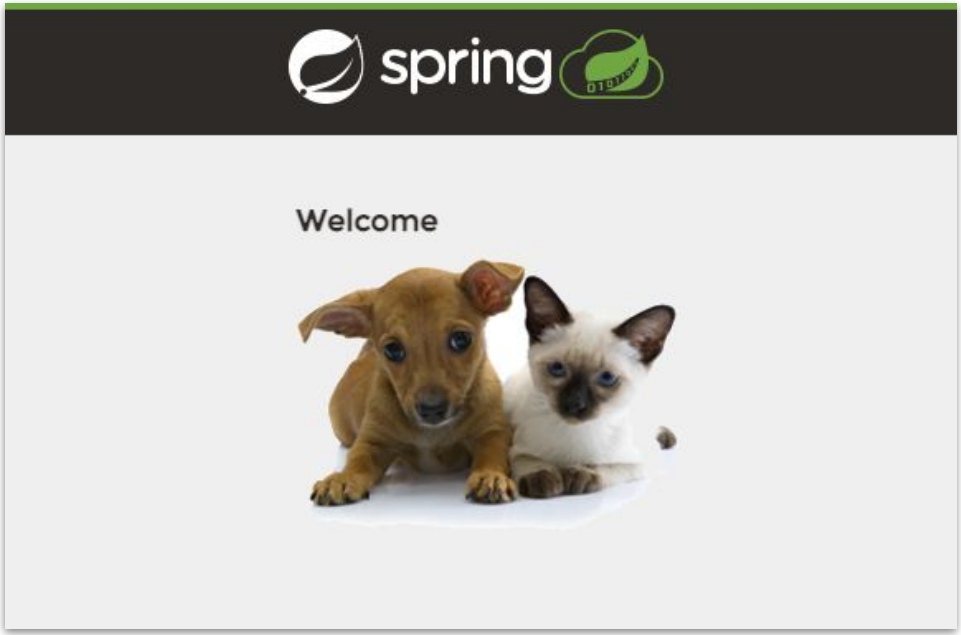
Weekly Workshops

https://www.datastax.com/workshops



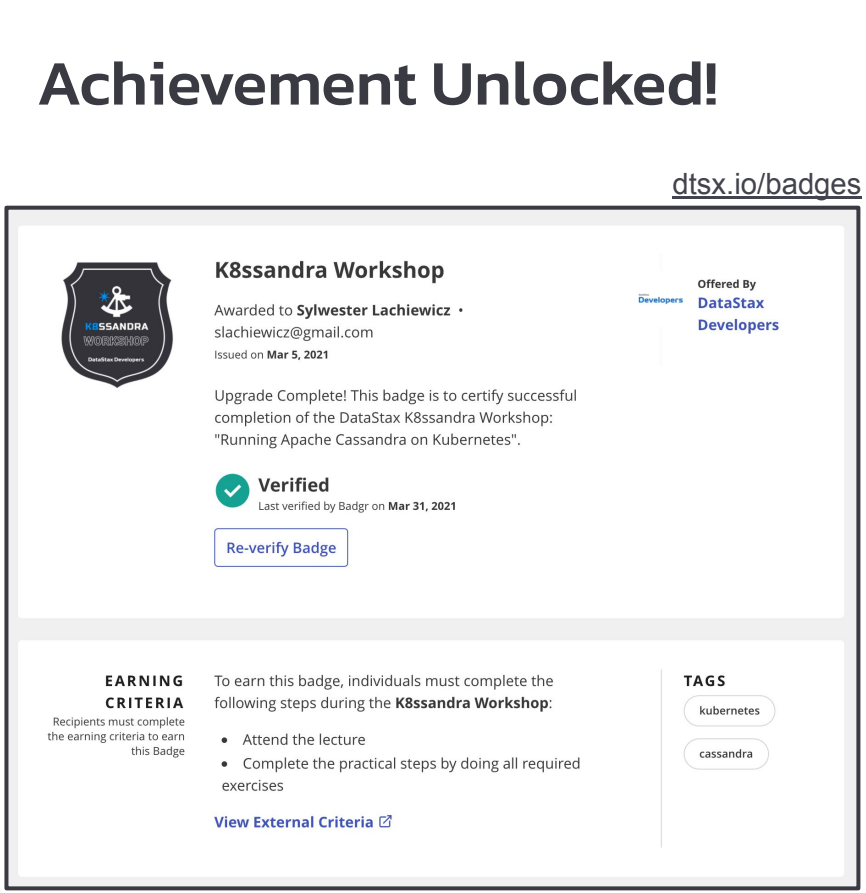
Housekeeping

App: spring-petclinic.github.io



spring by Pivotal.

Achievement Unlocked!



Aron L.
Marcin Brzozowski
Demre Buyuk
Muthu Krishnan
Arvind V.
Parth Trambadiya
Jasbir Singh

Prateek Jain
Roozbeh Dargahi
Andrey Deryabin
Akshay Wakhare
Pranav Anant
Joshi
Haris
Juan Alonso

Santosh Nepali
Jorge Ortiz
Ankit Bhavsar
Aneliya Klevleeva
Martin Coronel
Govindasamy
Ville Kerminen
Priya Jakhar

Paul Robu
Avinash Upadhyaya
Włodzimierz Kozłowski
Sharath Koushik
Tom Rota
Joel Reis
Francesco Abbate

Sylwester Lachiewicz
Ankit Bhavsar
Jasbir Singh

AND MANY OTHERS!

menti.com


Apache
CASSANDRA
25th
DS COMICS
DATASTAX

Go to www.menti.com and use the code 3491 9972
Inequality predicates are allowed on ...
4
3
5
1
All table columns
Partition key columns
clustering key columns
No inequality predicates are allowed
2:10:19 / 2:26:05

Go to www.menti.com and use the code 3491 9972
Leaderboard
4821 p spanda
4820 p Agent X9
fastest
4775 p Sam
4711 p CCedrickThePresenter
4468 p shubham
4371 p aaa
3895 p vignesh
3877 p adry
3861 p Millie
3812 p Puggie
2:11:07 / 2:26:05

HandsOn #1 Create Astra

DataStax Astra

Get your instance here:

- dtsx.io/workshop

GitHub

Repository:

- dtsx.io/sql-to-nosql

DataStax Developers

Agenda

01

Why switch from SQL to NoSQL?

02

Three migration approaches

03

Data modeling SQL vs NoSQL

04

Migrate your data model

05

Moving data with DSBulk

06

Resources

Agenda

01

Why switch from SQL to NoSQL?

02

Three migration approaches

03

Data modeling SQL vs NoSQL

04

Migrate your data model

05

Moving data with DSBulk

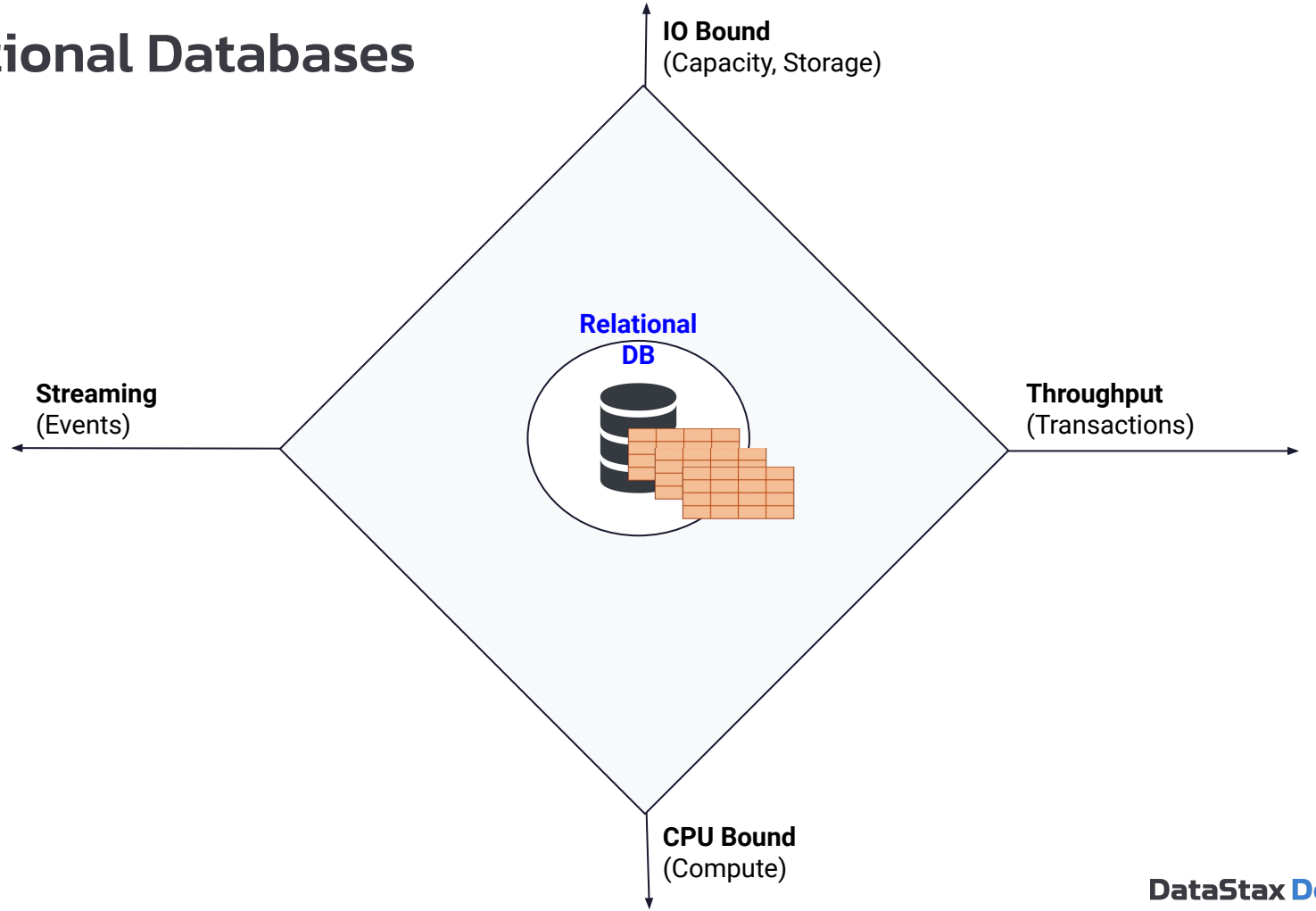
06

Resources

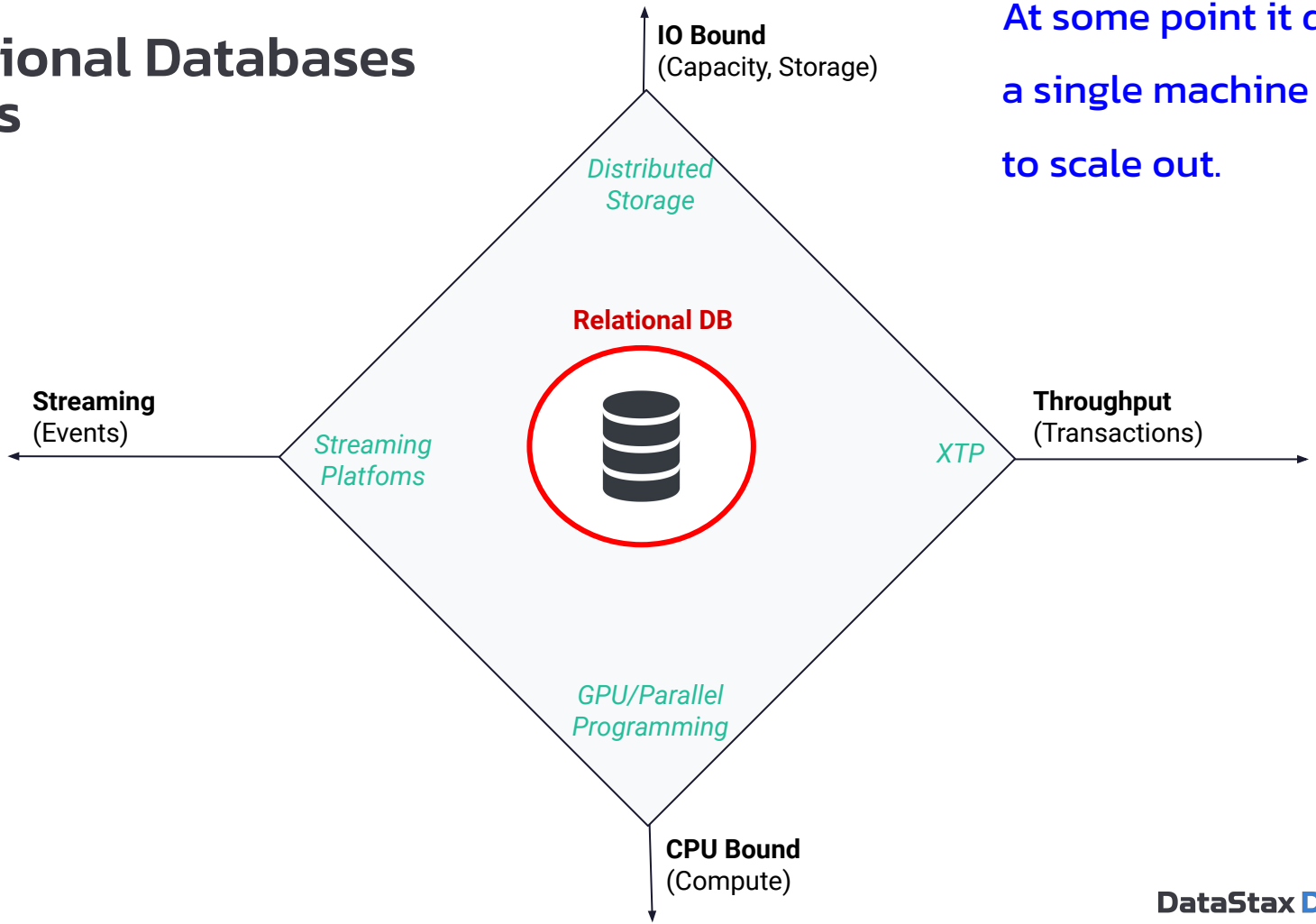
Why switch from SQL to NoSQL?

Performance, scale, and resiliency

Relational Databases

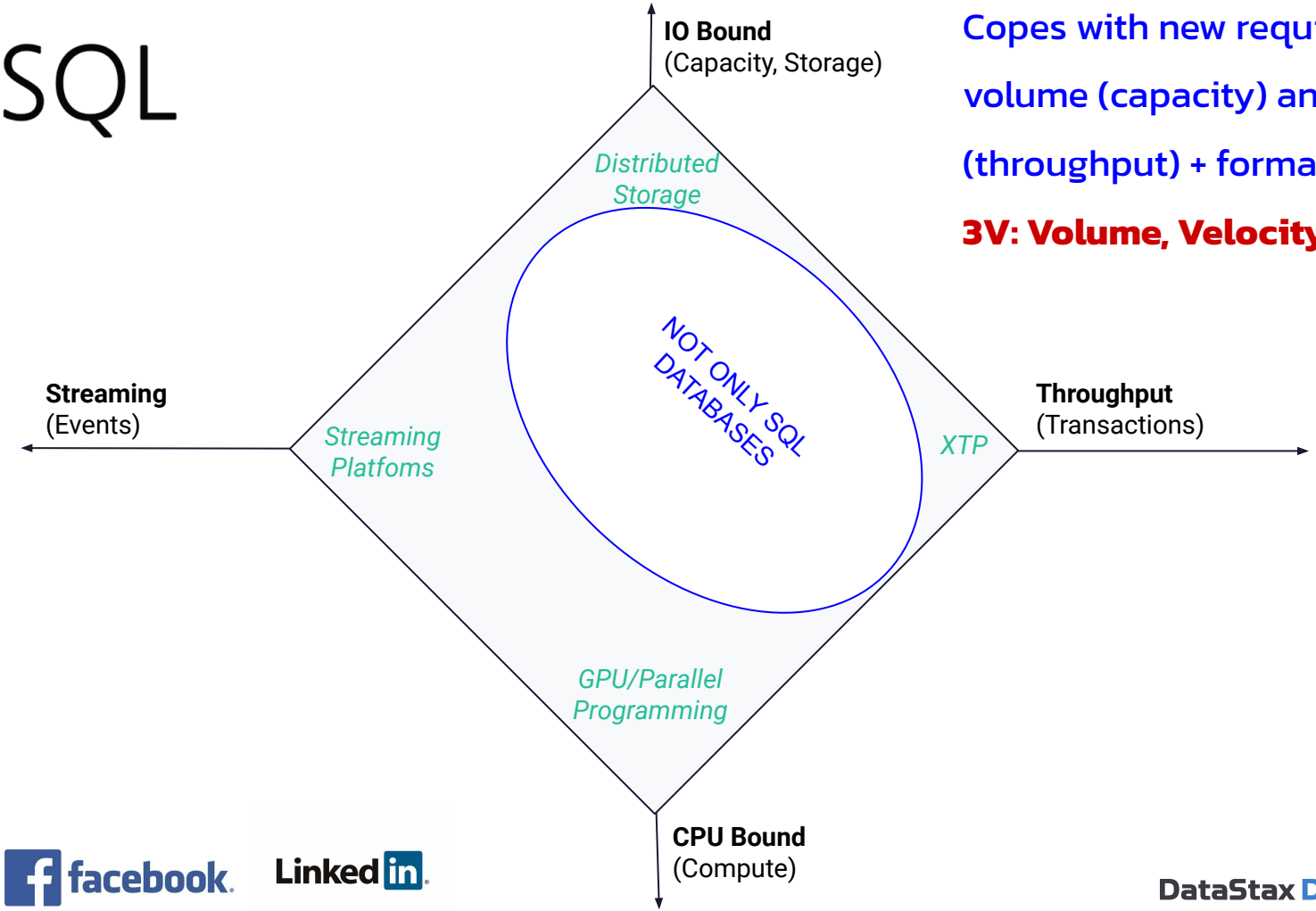


Relational Databases Limits



At some point it does not fit a single machine you need to scale out.

Not Only SQL



Copes with new requirements in volume (capacity) and velocity (throughput) + format (variety)

3V: Volume, Velocity, Variety

Google facebook LinkedIn

Agenda

01

Why switch from SQL to NoSQL?

02

Three migration approaches

03

Data modeling SQL vs NoSQL

04

Migrate your data model

05

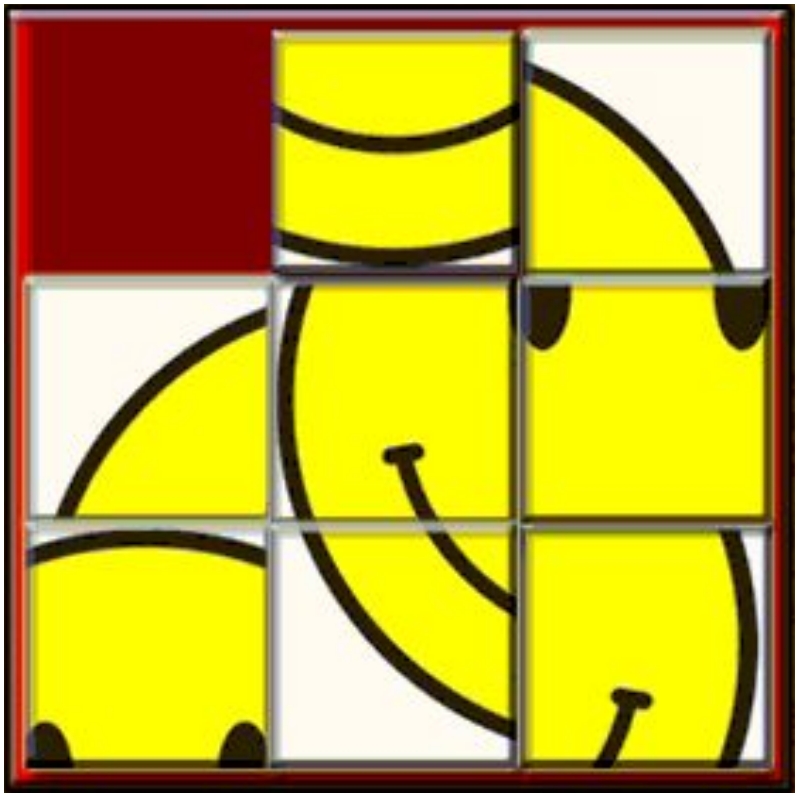
Moving data with DSBulk

06

Resources

42 application & data migration approaches

Let’s get started!!! (this is going to take a while)



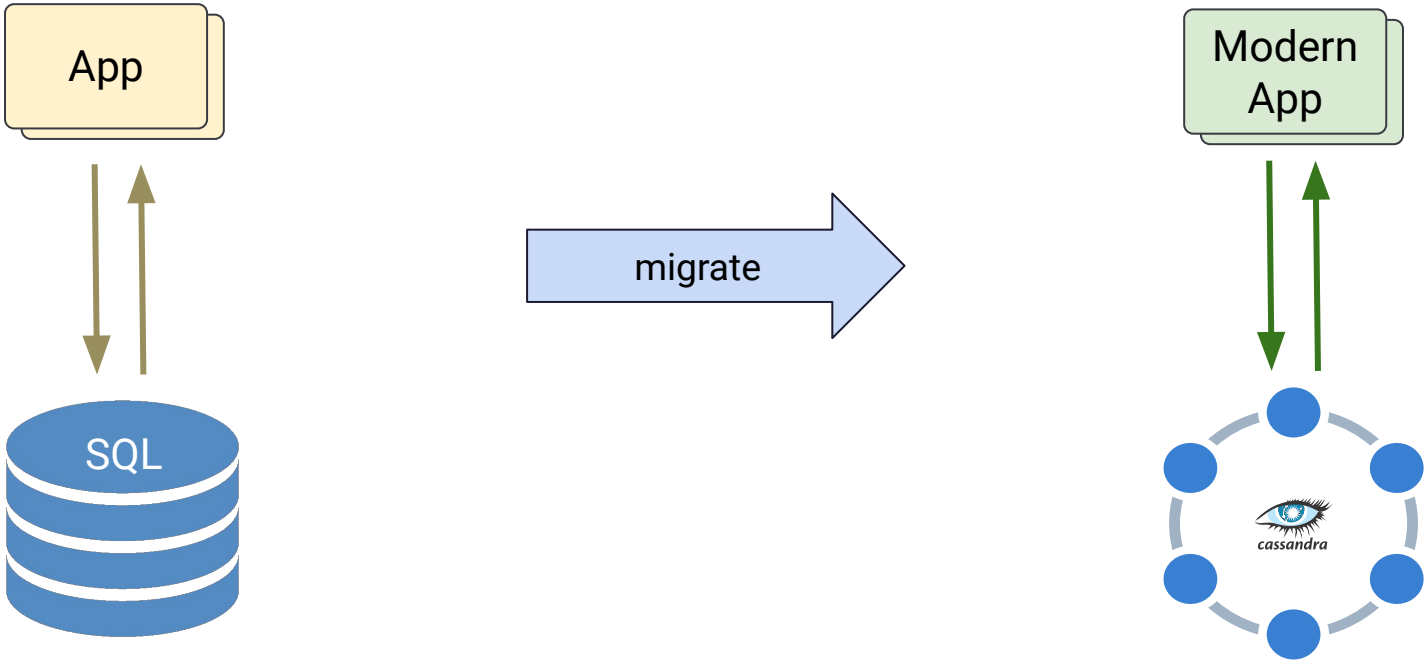
Migrating from SQL To NoSQL can sometimes feel like…

...one of those slide puzzles where you have to reason out the order of operations to get it right.

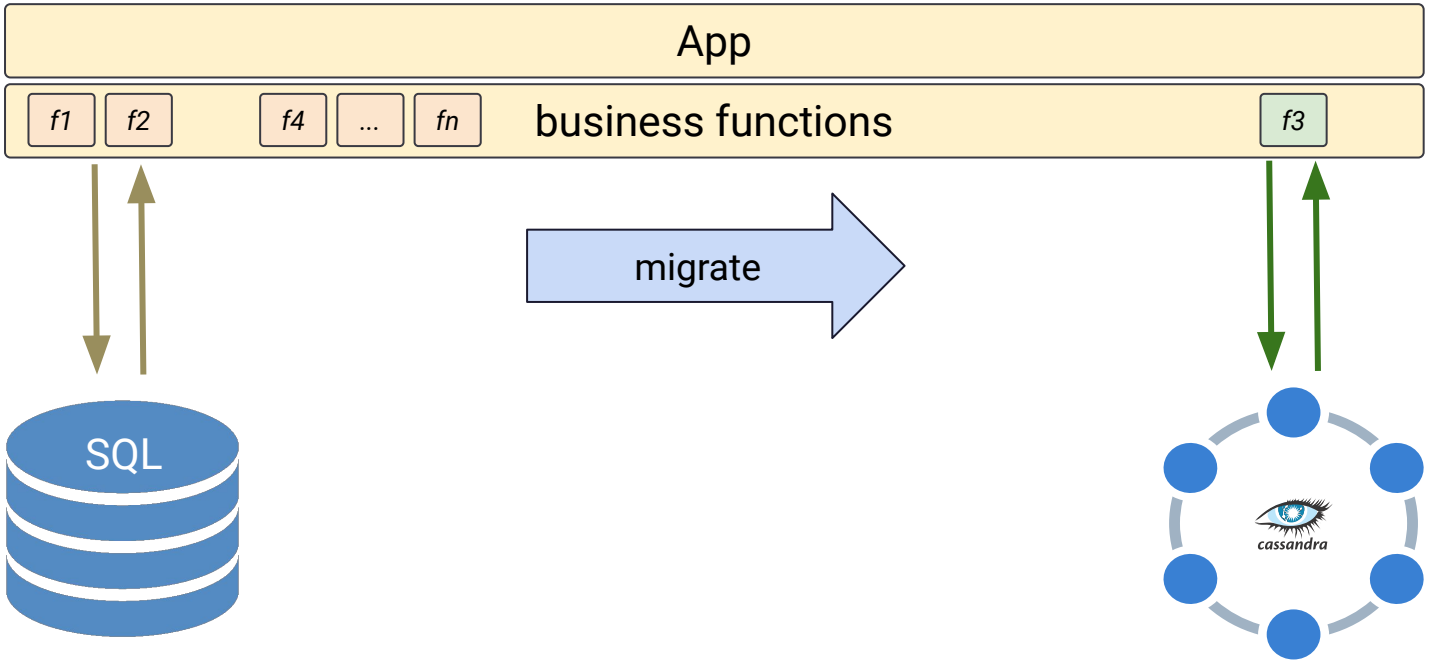
Our goal is to help you solve for smiley face.

The problem definition

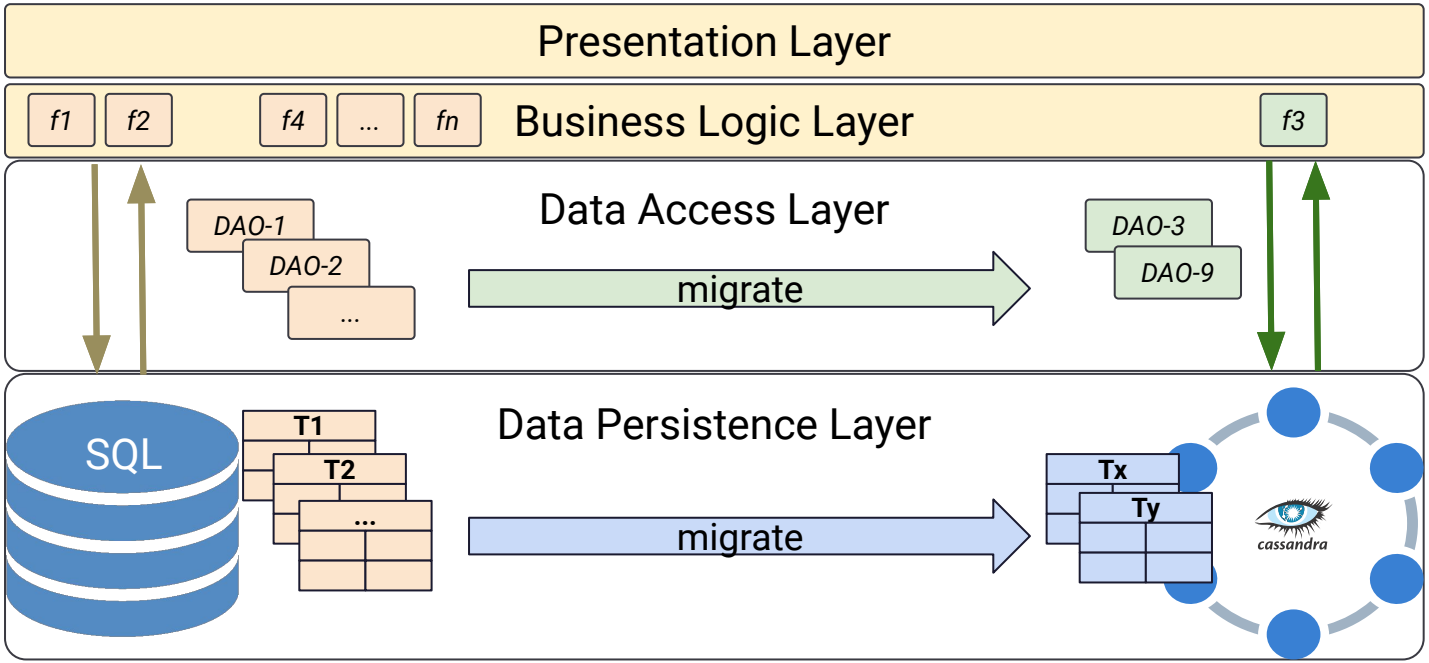
The Migration Problem Definition in an Ideal World



The Migration Problem Definition in the Real World



The Migration Problem Definition Details



The three migration approaches

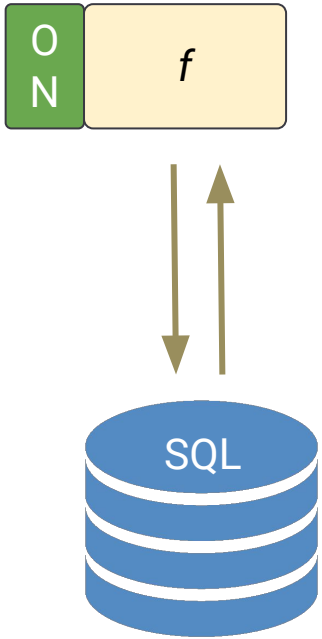
The Three Approaches

1. Offline Migration
2. Zero-Downtime Migration with Shadow Writes
3. Minimal-Downtime Migration with Dual Reads

Approach 1: Offline Migration

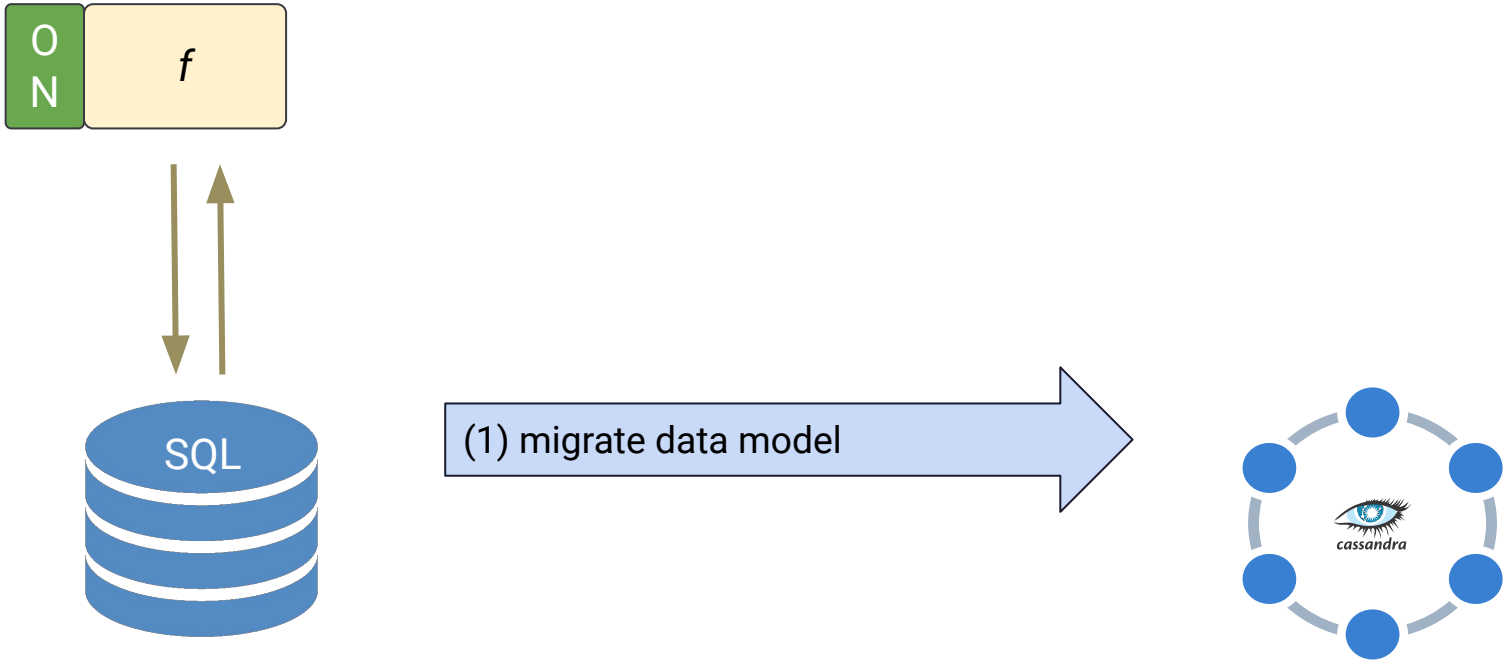
Offline Migration

Approach 1



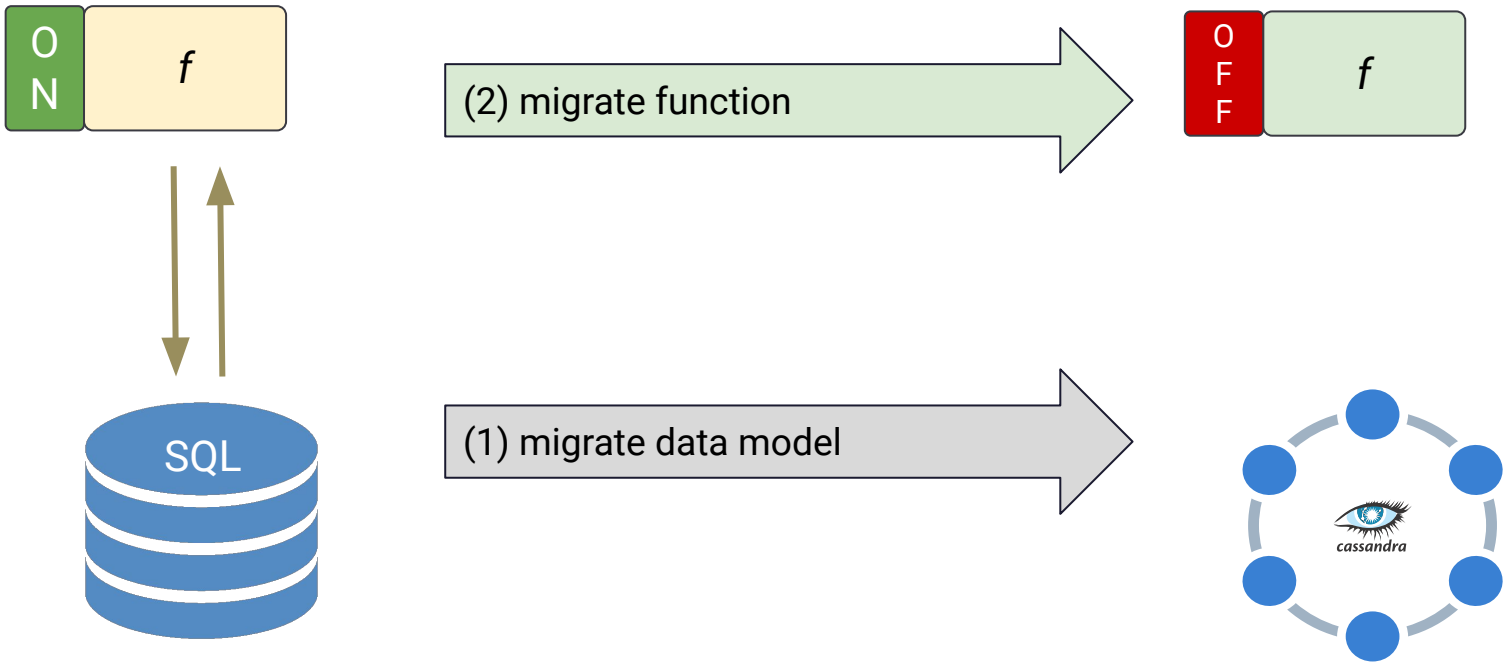
Offline Migration

Approach 1



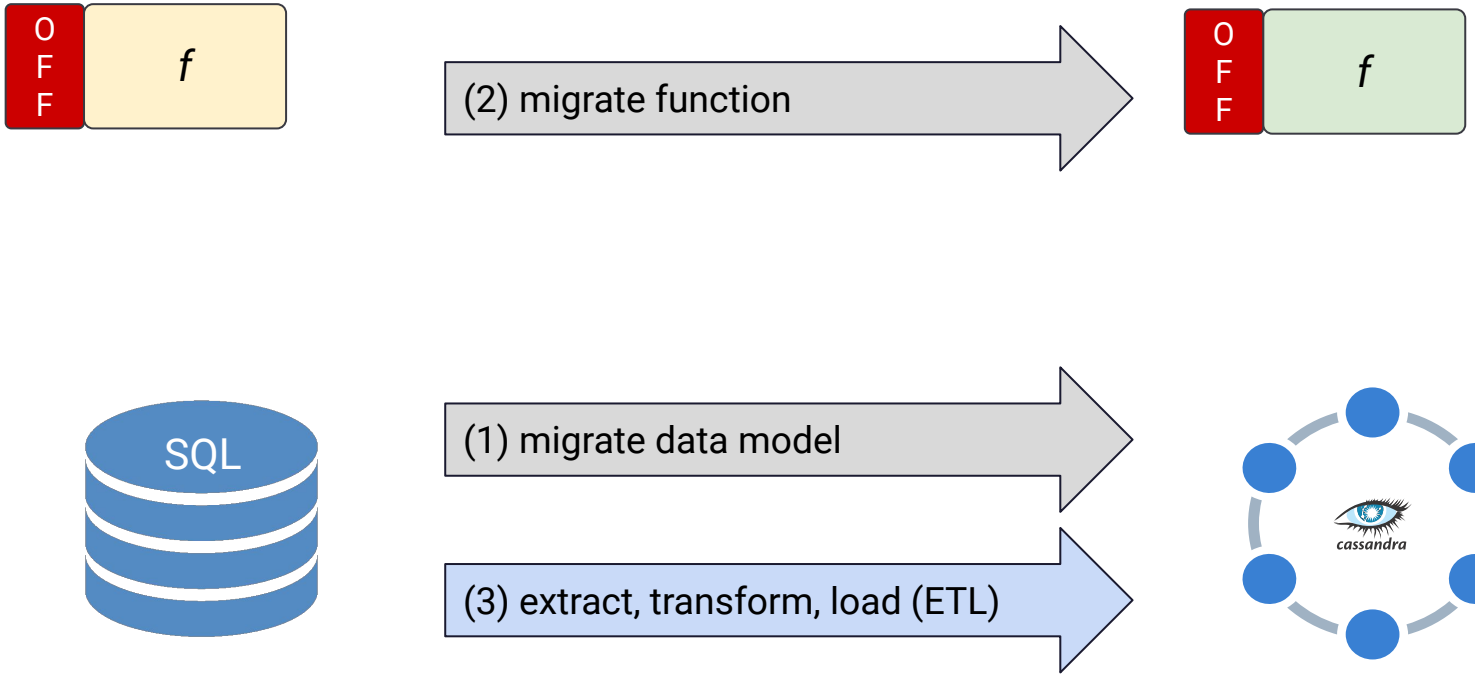
Offline Migration

Approach 1



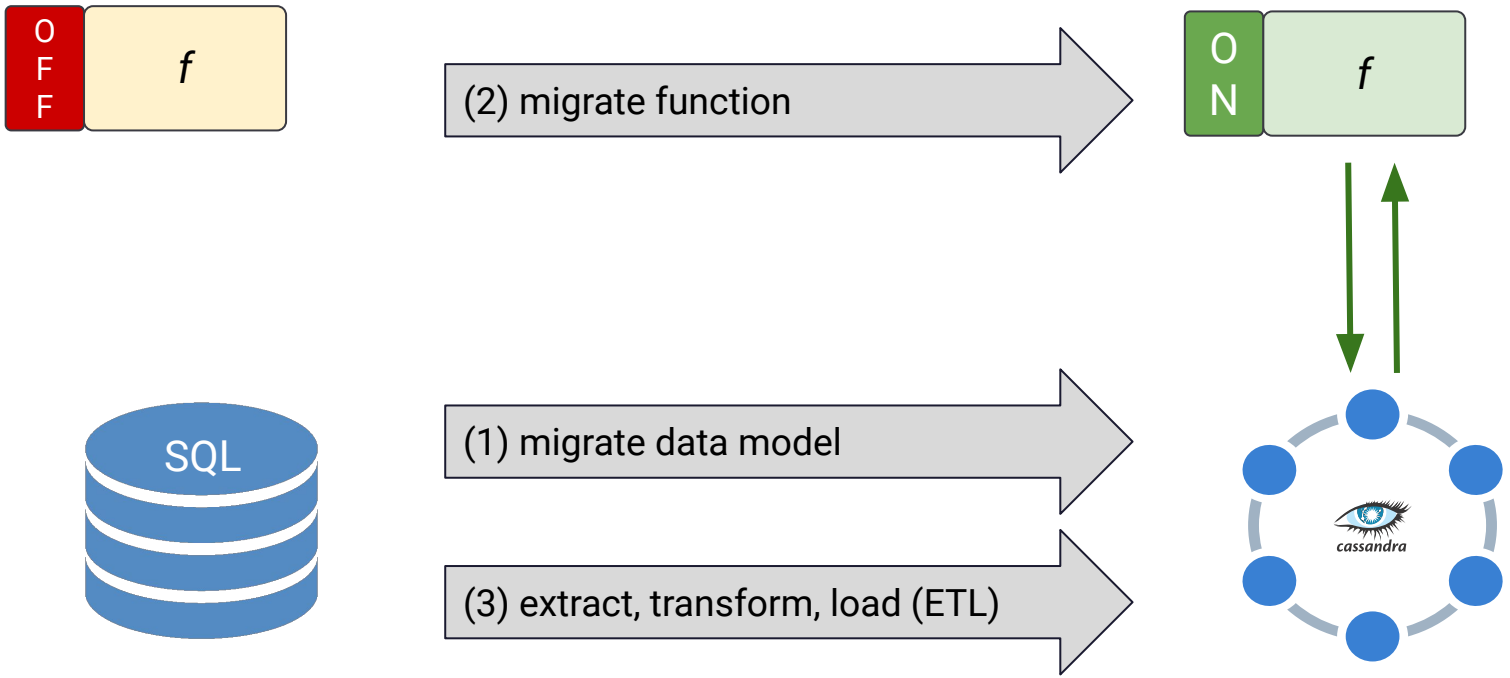
Offline Migration

Approach 1



Offline Migration

Approach 1

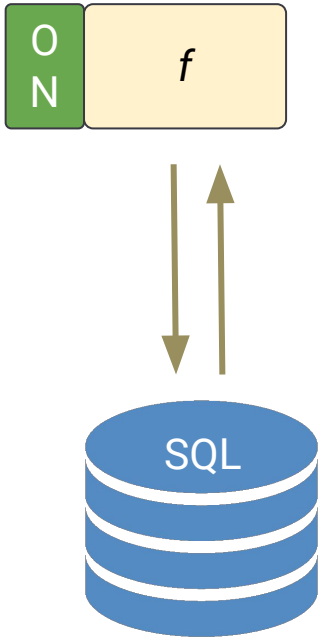


Approach 2: Zero-Downtime Migration with Shadow Writes

Shadow writes

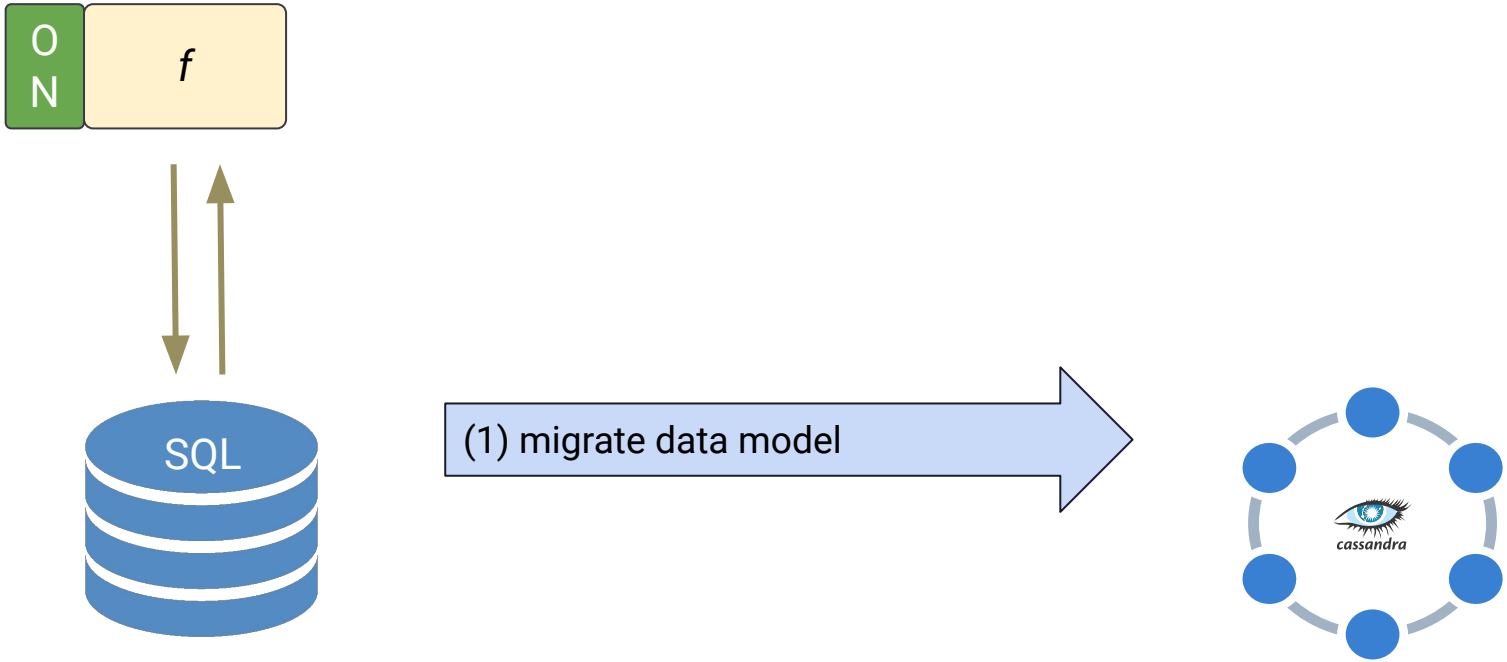
Zero-Downtime Migration with Shadow Writes

Approach 2



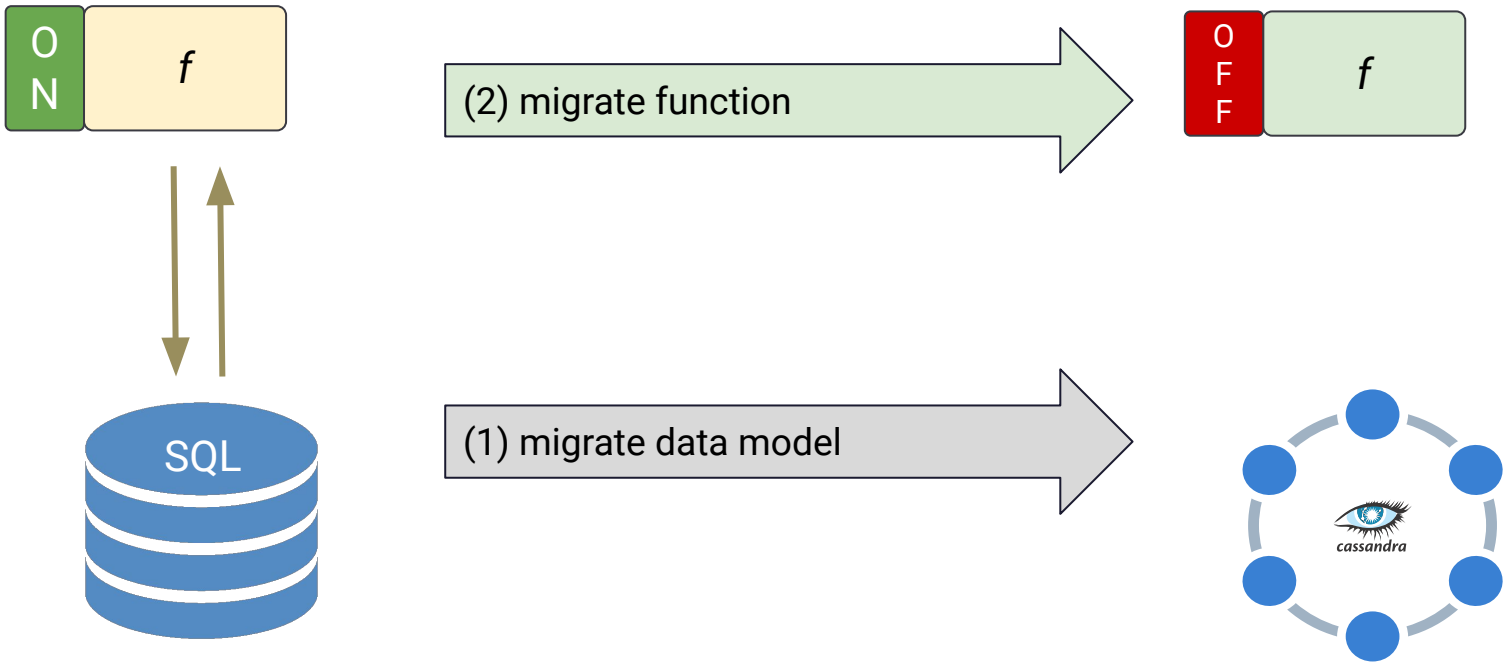
Zero-Downtime Migration with Shadow Writes

Approach 2



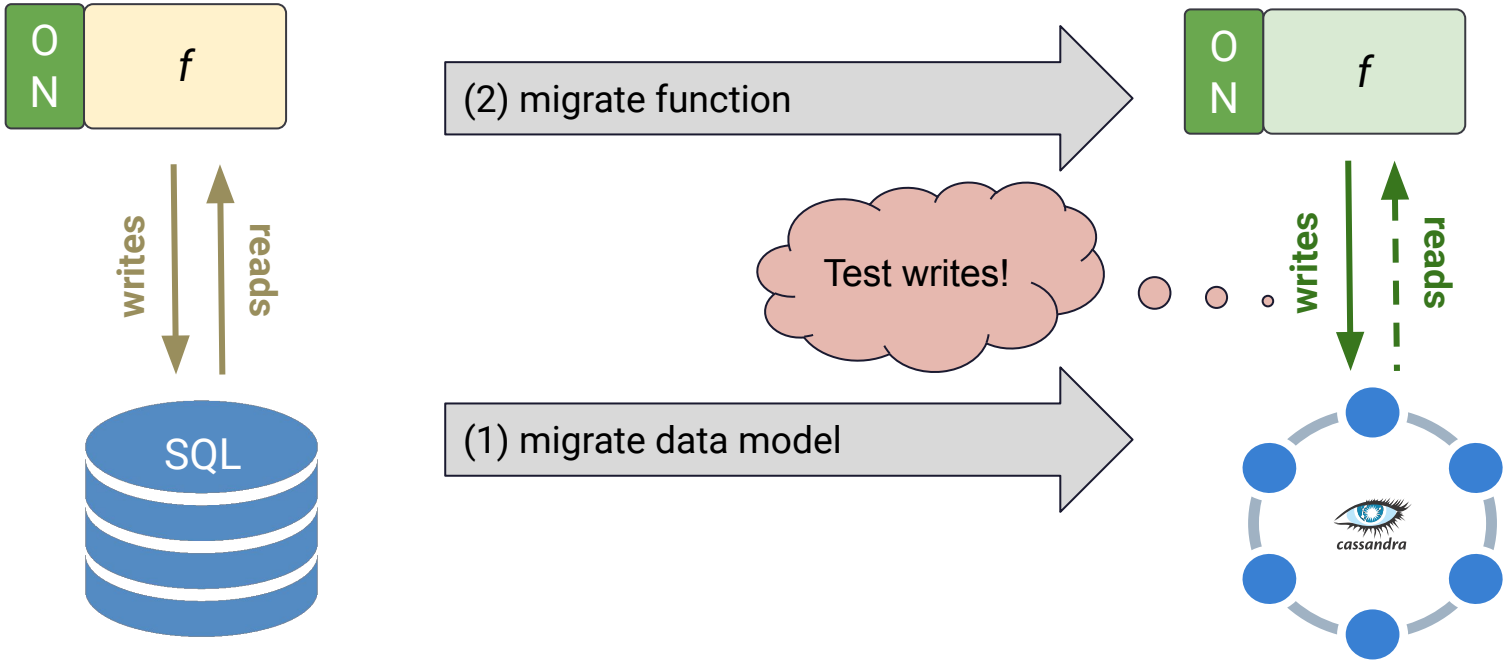
Zero-Downtime Migration with Shadow Writes

Approach 2



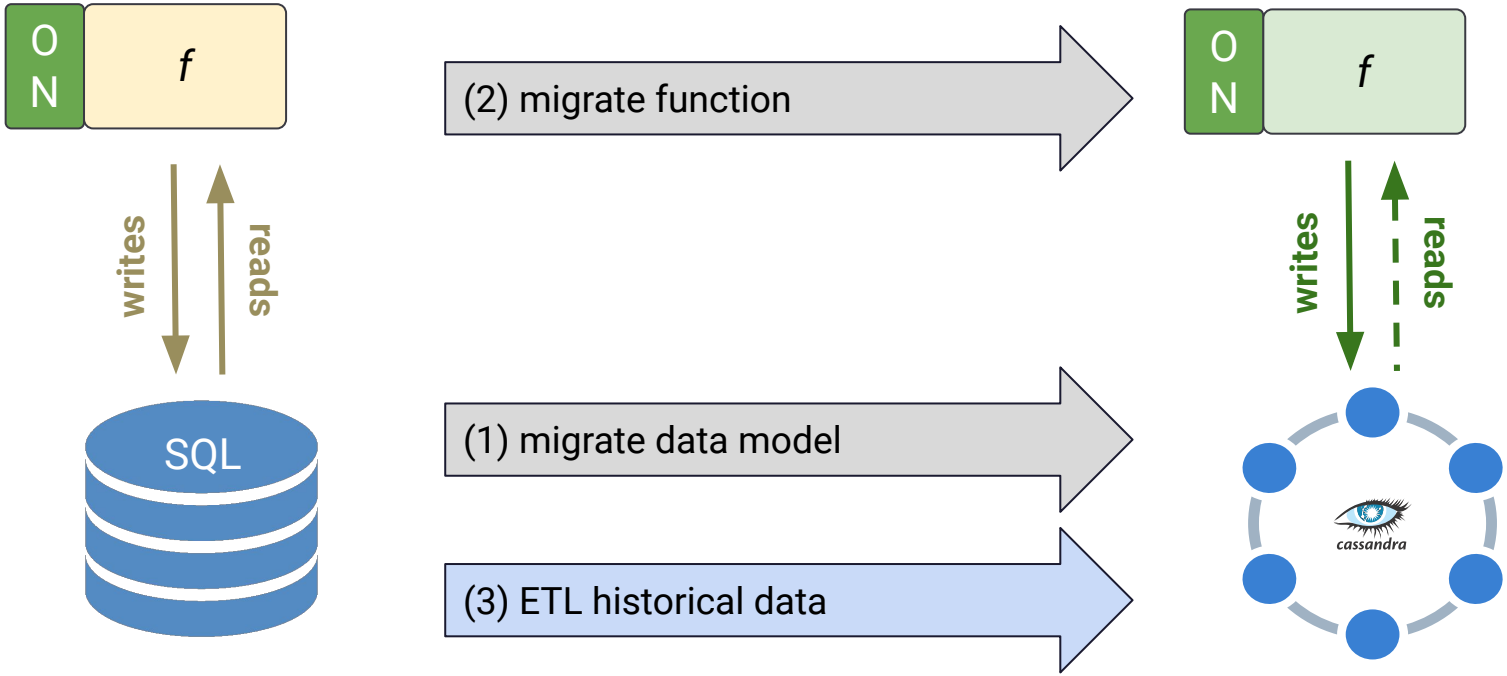
Zero-Downtime Migration with Shadow Writes

Approach 2



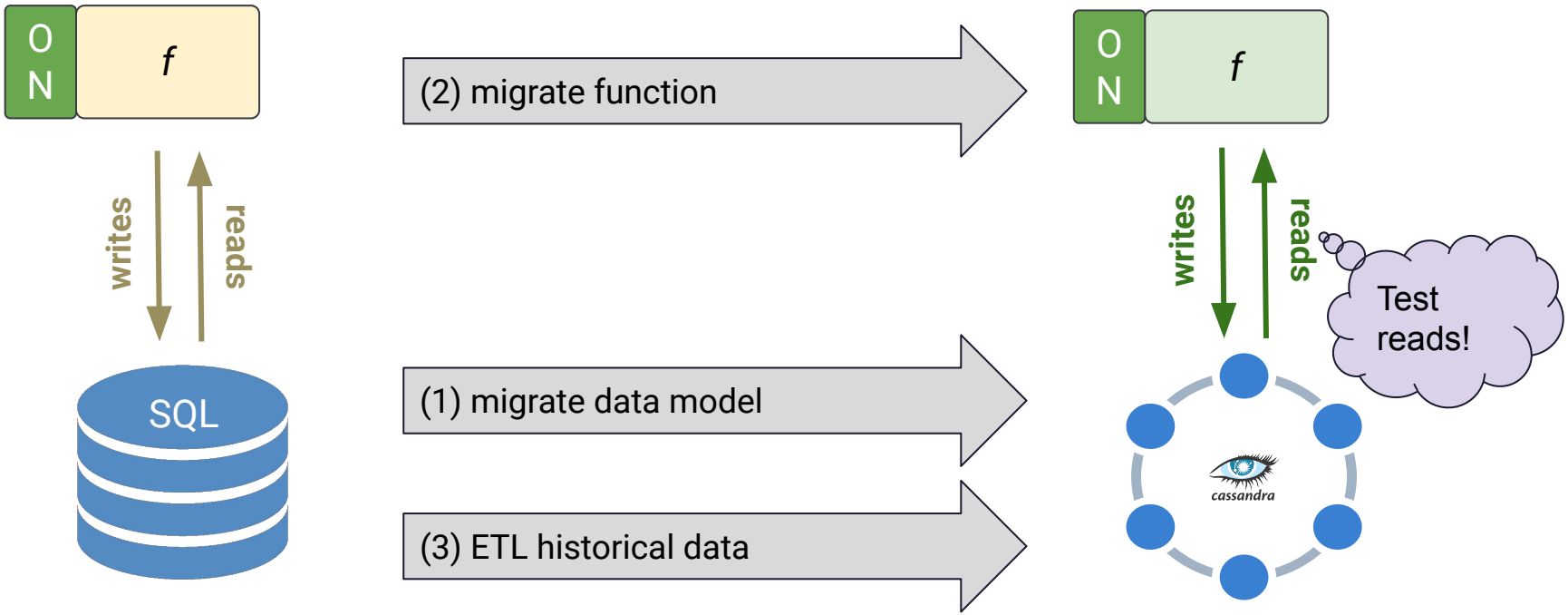
Zero-Downtime Migration with Shadow Writes

Approach 2



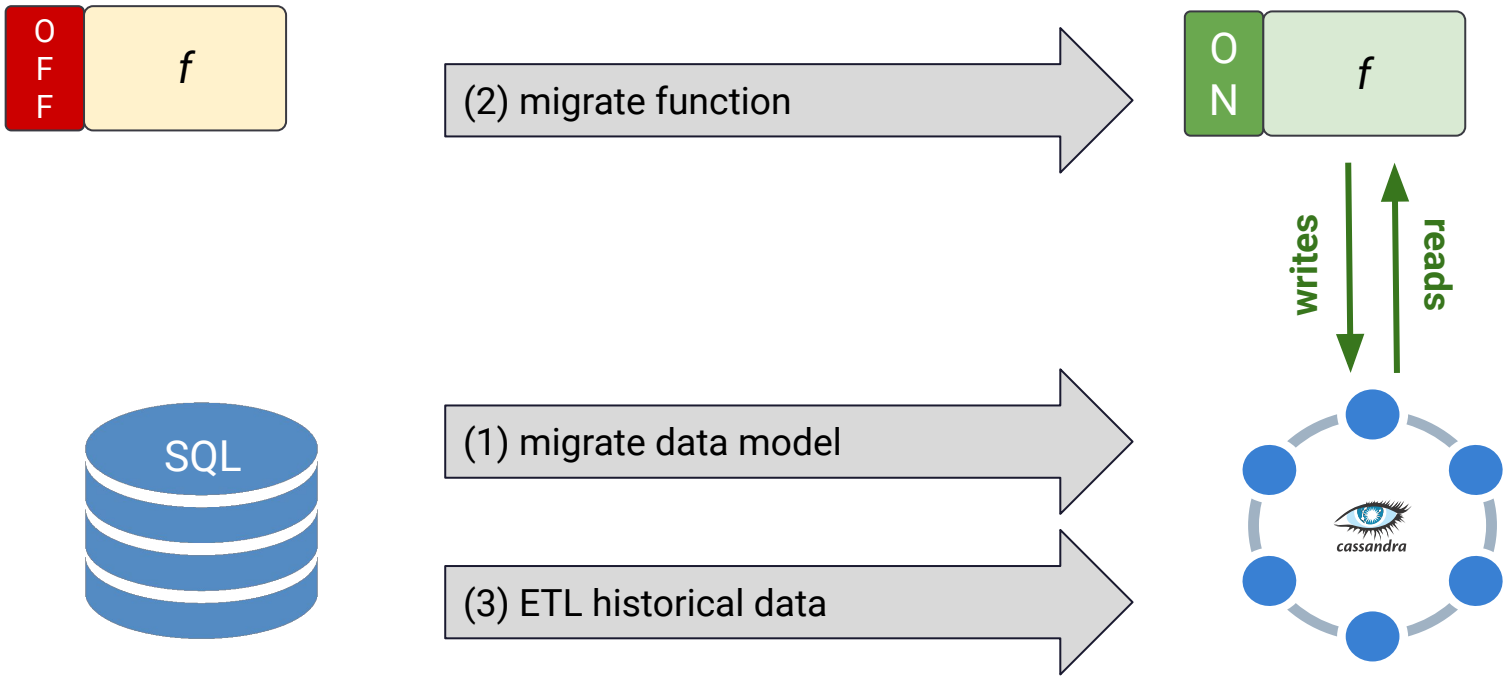
Zero-Downtime Migration with Shadow Writes

Approach 2



Zero-Downtime Migration with Shadow Writes

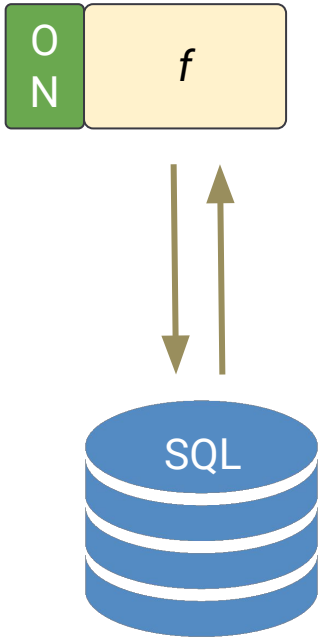
Approach 2



Approach 3: Minimal-Downtime Migration with Dual Reads

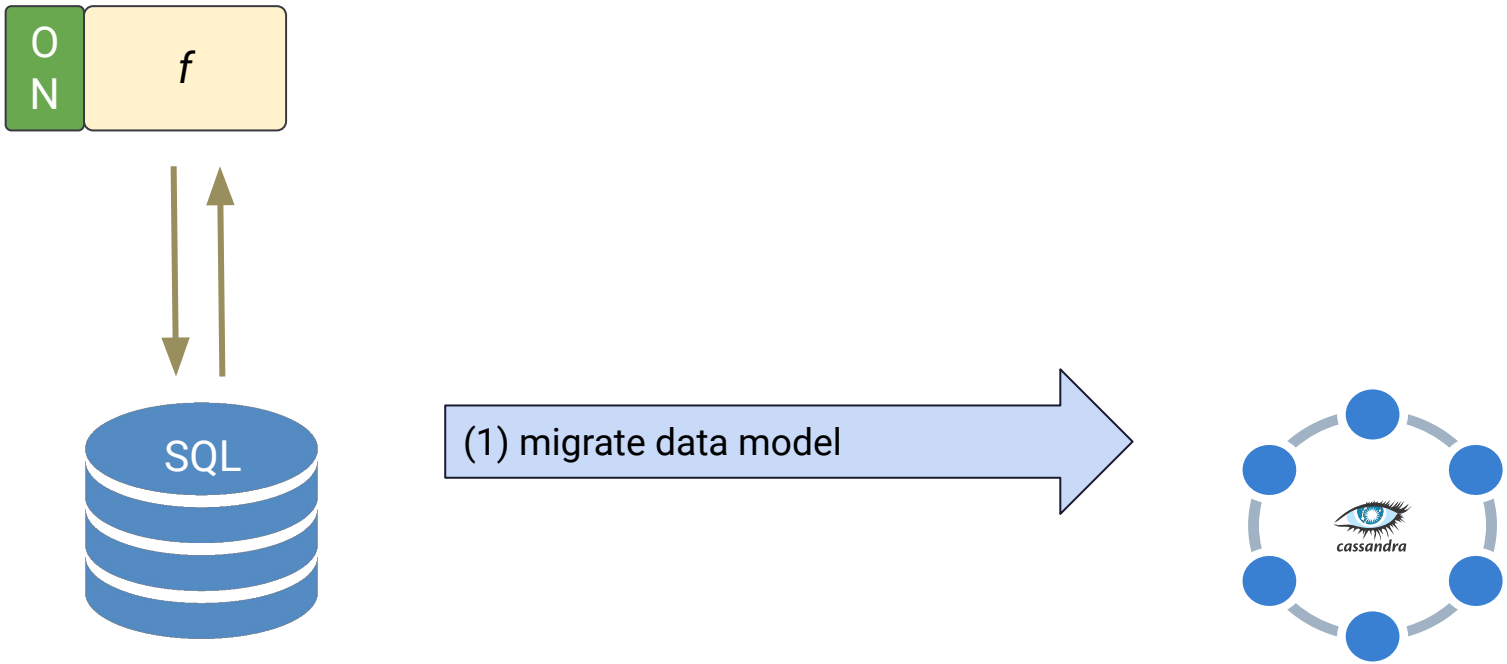
Minimal-Downtime Migration with Dual Reads

Approach 3



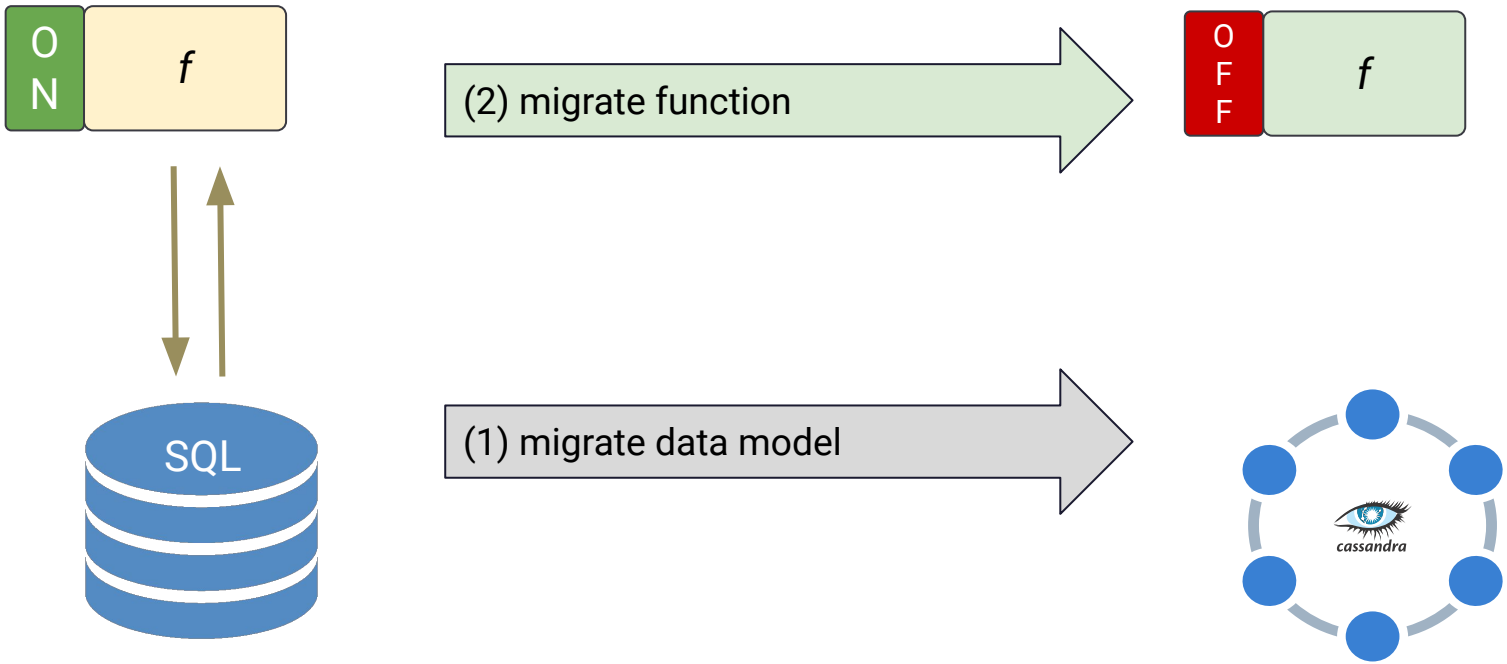
Minimal-Downtime Migration with Dual Reads

Approach 3



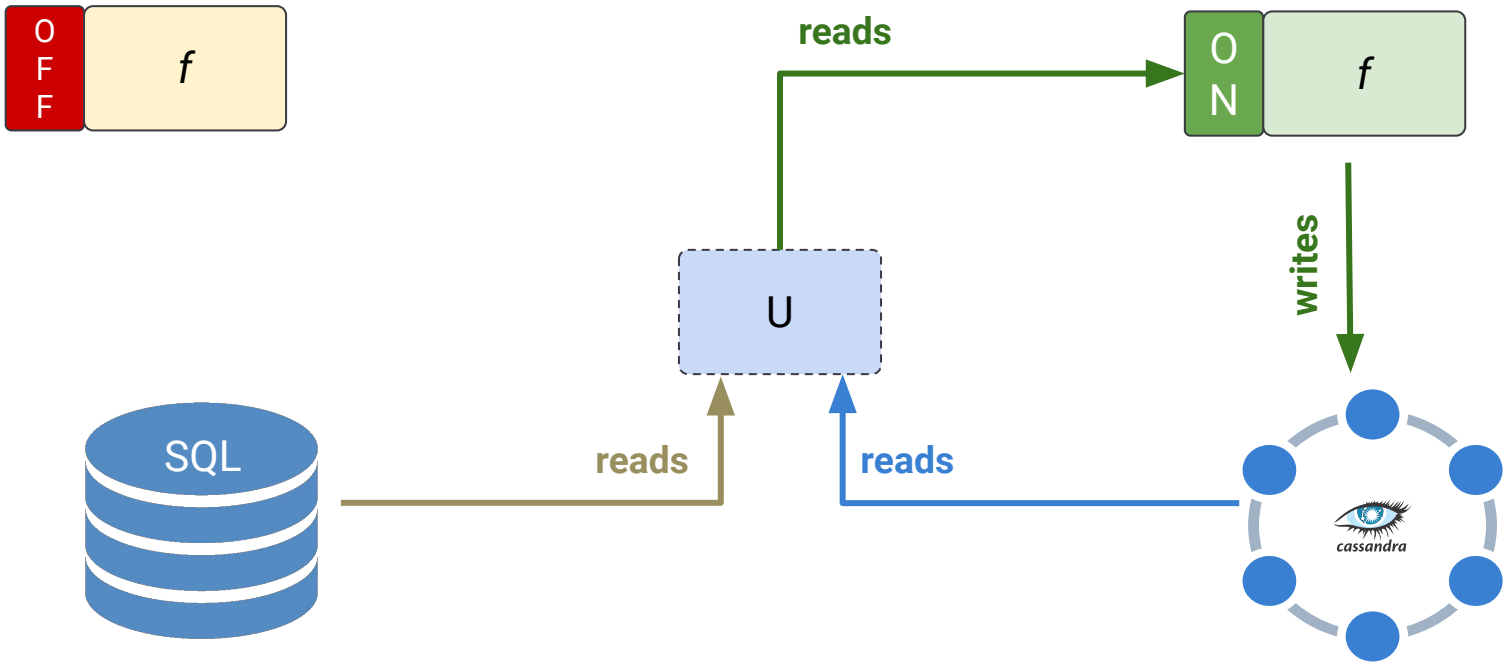
Minimal-Downtime Migration with Dual Reads

Approach 3



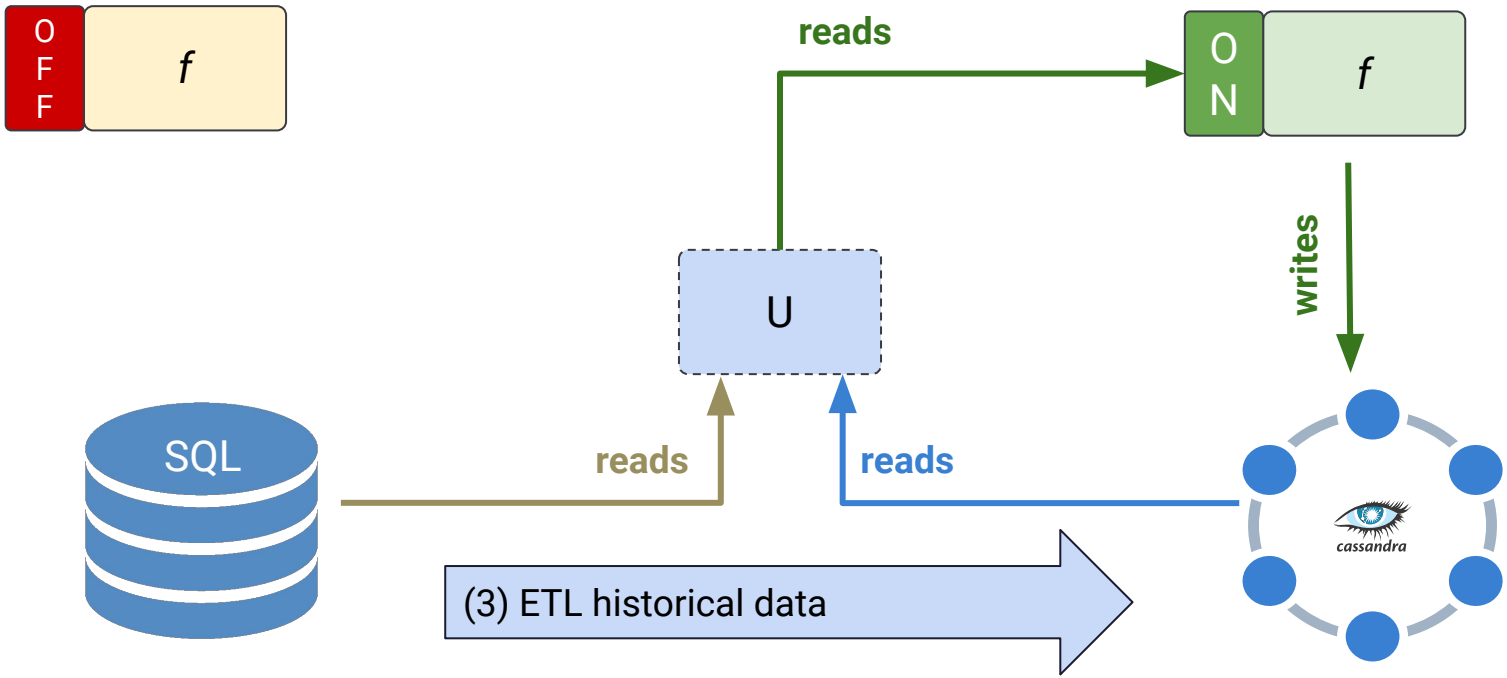
Minimal-Downtime Migration with Dual Reads

Approach 3



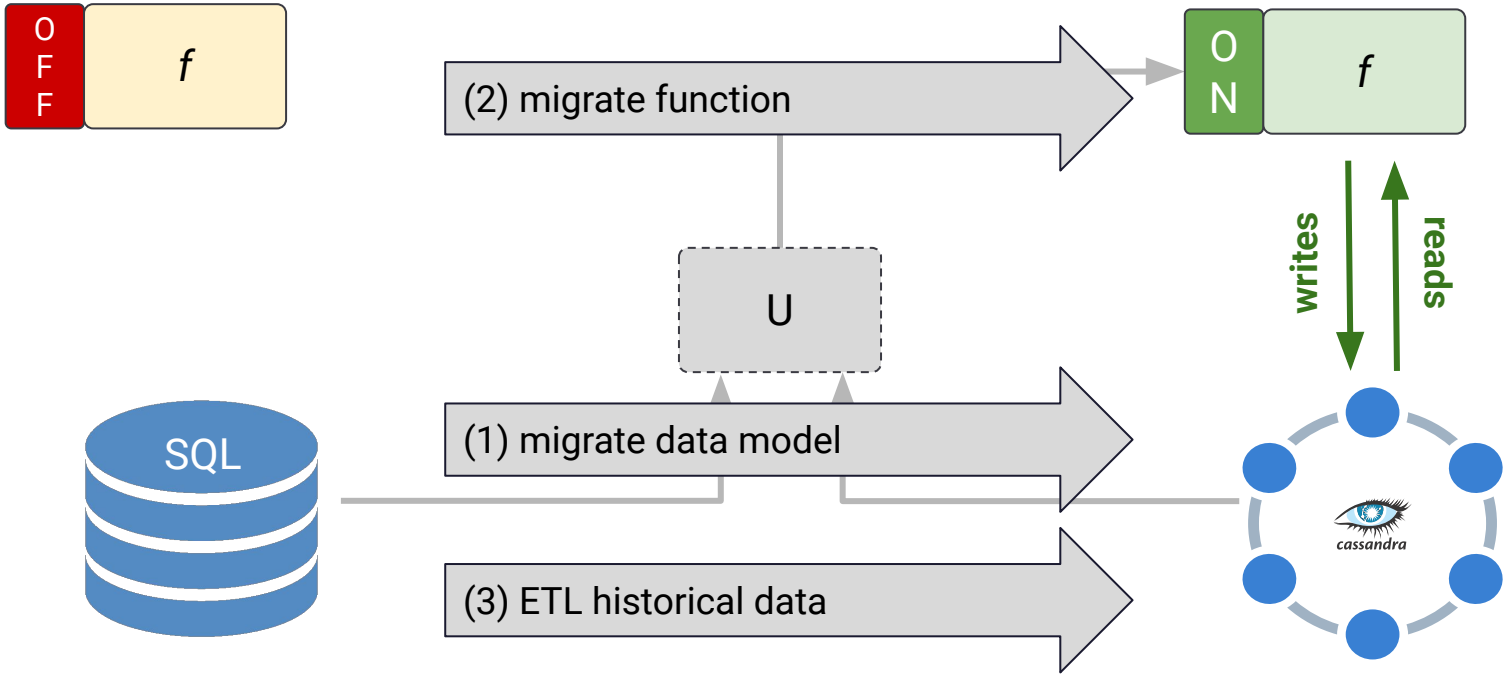
Minimal-Downtime Migration with Dual Reads

Approach 3



Minimal-Downtime Migration with Dual Writes

Approach 3



Common Tasks and Challenges

- Migrate data model
  - SQL vs NoSQL data modeling
  - Query-driven table schema design
  - Denormalization, data duplication, data nesting
- Migrate function
  - Business logic does not change
  - Focus on CRUD operations, DAO and microservices
- Migrate data
  - Shadowed traffic, service proxies
  - ETL, Import/export utilities, DSBulk, Spark

Agenda

01

Why switch from SQL to NoSQL?

02

Three migration approaches

03

Data modeling SQL vs NoSQL

04

Migrate your data model

05

Moving data with DSBulk

06

Resources

Data modeling SQL vs NoSQL

“This is the way” - mando

Normalization

"Database normalization is the process of structuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. It was first proposed by Edgar F. Codd as part of his relational model."

PROS: Simple write, Data Integrity
CONS: Slow read, Complex Queries

Employees

userId	deptId	firstName	lastName
1	1	Edgar	Codd
2	1	Raymond	Boyce

Departments

departmentId	department
1	Engineering
2	Math

Denormalization

"Denormalization is a strategy used on a database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data"

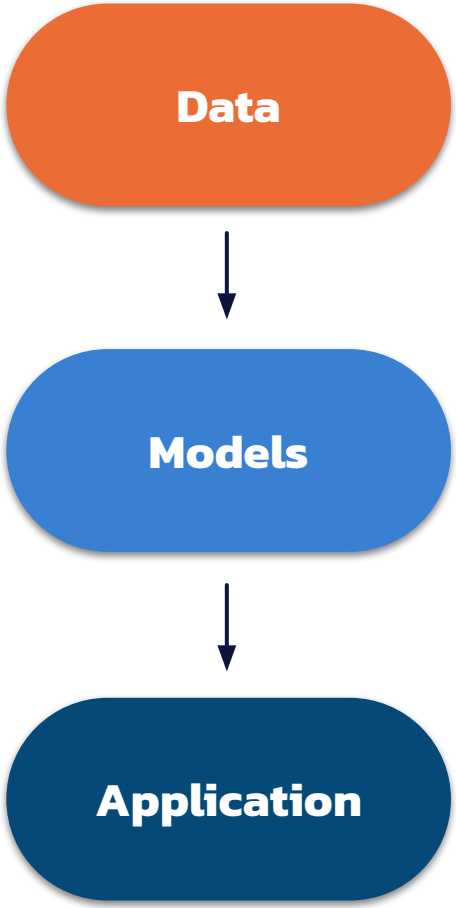
PROS: Quick Read, Simple Queries
CONS: Multiple Writes, Manual Integrity

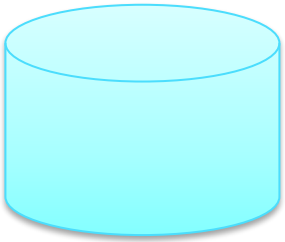
Employees

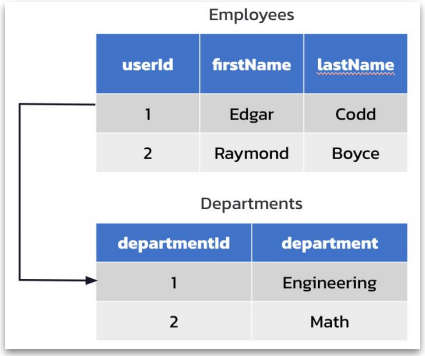
userId	firstName	lastName	department
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Engineering
3	Sage	Lahja	Math
4	Juniper	Jones	Botany

Relational Data Modelling

1. Analyze raw data
2. Identify entities, their properties and relations
3. Design tables, using **normalization** and foreign keys.
4. Use JOIN when doing queries to join normalized data from multiple tables

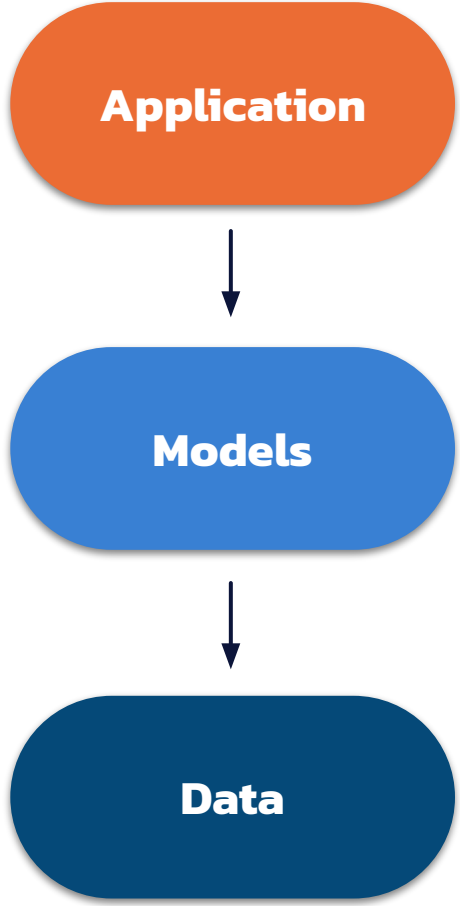






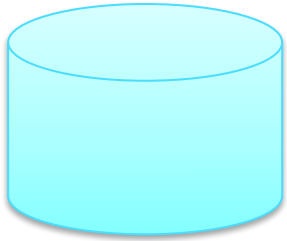
NoSQL Data Modelling

1. Analyze user behaviour (customer first!)
2. Identify workflows, their dependencies and needs
3. Define Queries to fulfill these workflows
4. Knowing the queries, design tables, using **denormalization**.
5. Insert and update multiple copies of data that may have resulted due to denormalization

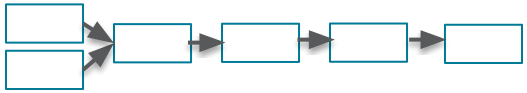


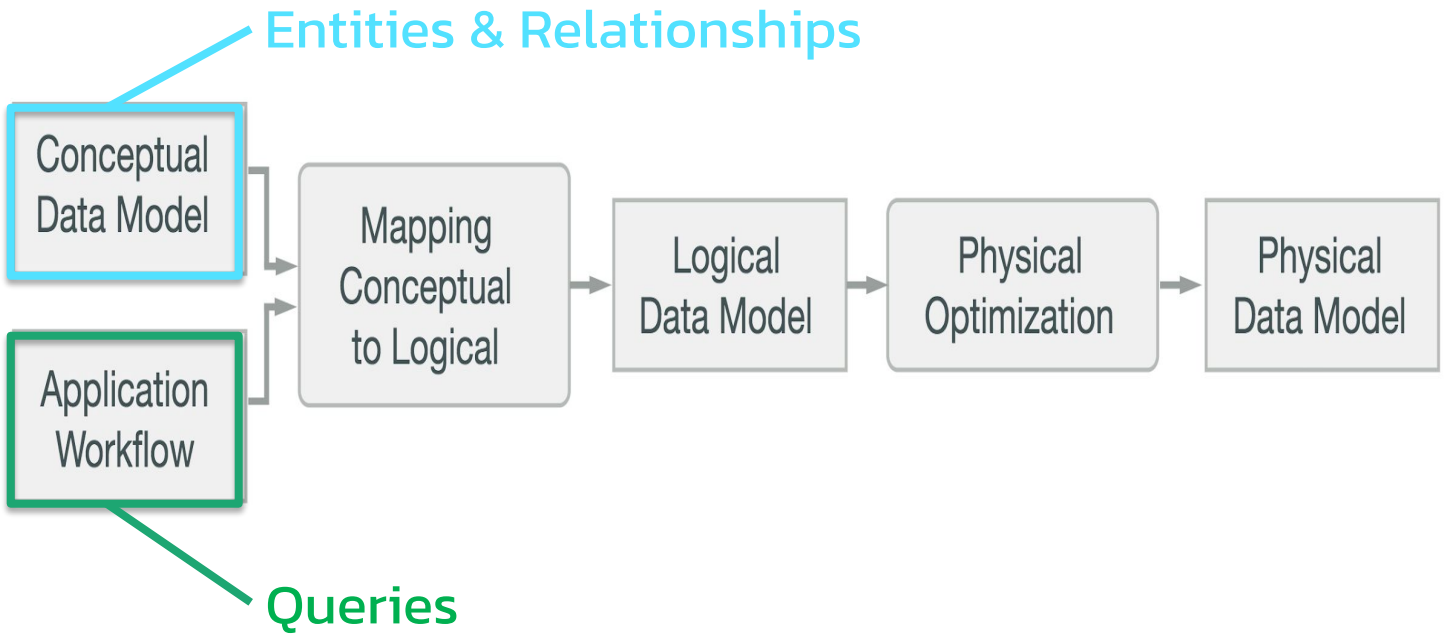
Employees

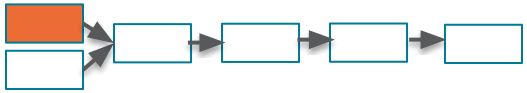
userId	firstName	lastName	department
1	Edgar	Codd	Engineering
2	Raymond	Boyce	Math
3	Sage	Lahja	Math
4	Juniper	Jones	Botany

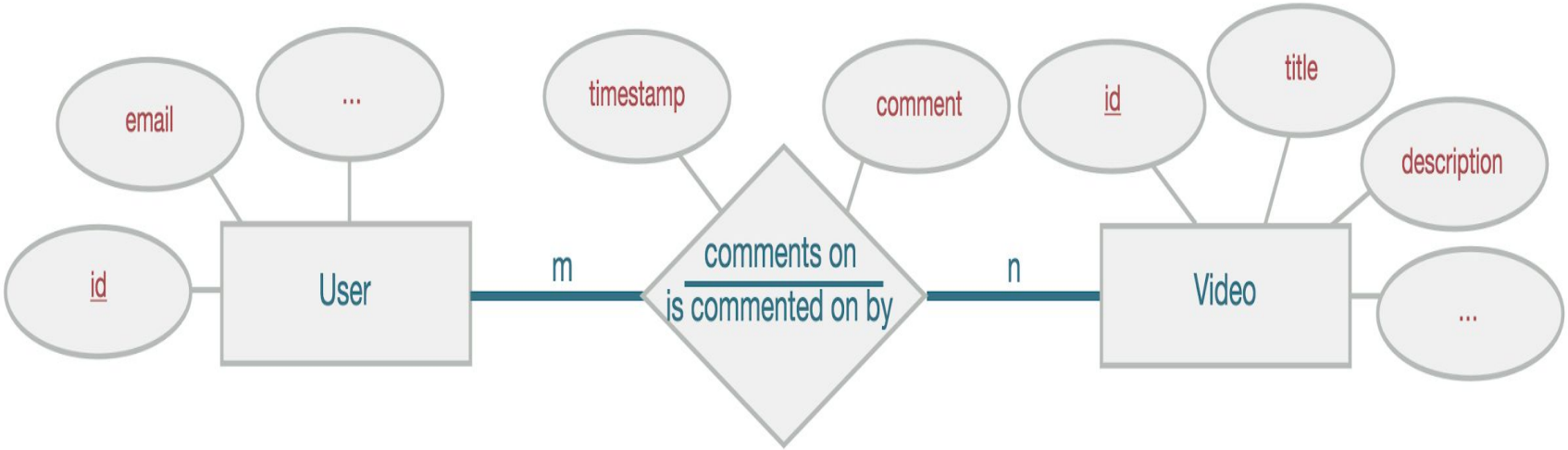


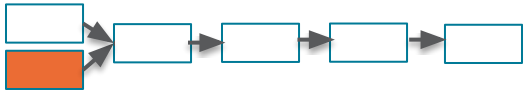

©2020 & TM Lucasfilm Ltd.
Disney+




Entities & Relationships
Conceptual Data Model
Application Workflow
Mapping Conceptual to Logical
Logical Data Model
Physical Optimization
Physical Data Model
Queries




email
...
id
User
m
timestamp
comment
comments on
is commented on by
n
id
title
description
Video
...

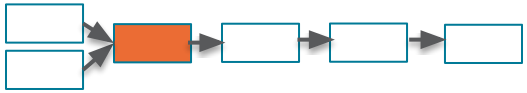


Use-Case I:
A User opens a Profile

WF1: Find comments related to target user using its identifier, most recent first

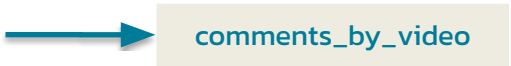
Use-Case II:
A User opens a Video Page

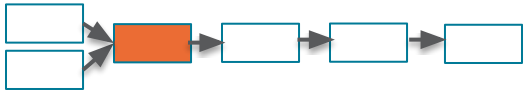
WF2: Find comments related to target video using its identifier, most recent first



Query I: Find comments posted for a user with a known id (show most recent first)

Query II: Find comments for a video with a known id (show most recent first)



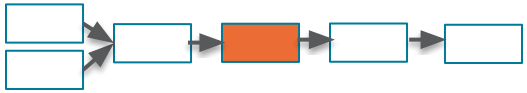


```
SELECT * FROM comments_by_user
WHERE userid = <some UUID>
```

comments_by_user

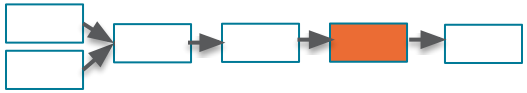
```
SELECT * FROM comments_by_video
WHERE videoid = <some UUID>
```

comments_by_video



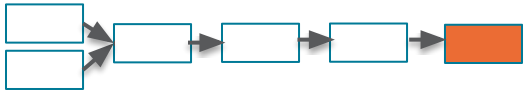
comments_by_user	
userid	K
creationdate	C ↓
commentid	C ↑
videoid	
comment	

comments_by_video	
videoid	K
creationdate	C ↓
commentid	C ↑
userid	
comment	



comments_by_user		
userid	UUID	K
commentid	TIMEUUID	C ↓
videoid	UUID	
comment	TEXT	

comments_by_video		
videoid	UUID	K
commentid	TIMEUUID	C ↓
userid	UUID	
comment	TEXT	



```
CREATE TABLE IF NOT EXISTS comments_by_user (
    userid uuid,
    commentid timeuuid,
    videoid uuid,
    comment text,
    PRIMARY KEY ((userid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);

CREATE TABLE IF NOT EXISTS comments_by_video (
    videoid   uuid,
    commentid timeuuid,
    userid    uuid,
    comment   text,
    PRIMARY KEY ((videoid), commentid)
) WITH CLUSTERING ORDER BY (commentid DESC);
```

Agenda

01

Why switch from SQL to NoSQL?

02

Three migration approaches

03

Data modeling SQL vs NoSQL

04

Migrate your data model

05

Moving data with DSBulk

06

Resources

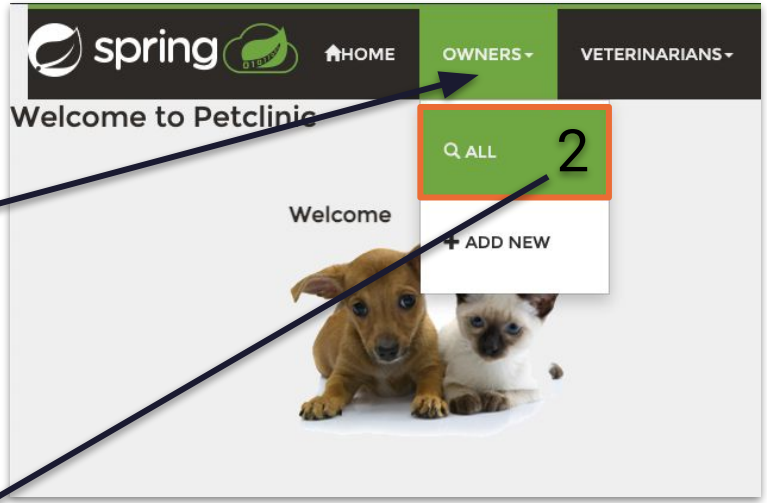
Migrate your data model

Data modeling is both art and science

Owners

Who actually owns who here?

Owner – Application View



Owners

Name	Address	City	Telephone	Pets
Heather Jones-Gilardi	AnotherPlace Dr.	Orlando	987654321	
David Gilardi	Nowhere St.	Orlando	123456789	Carrabba Shocker

Add Owner

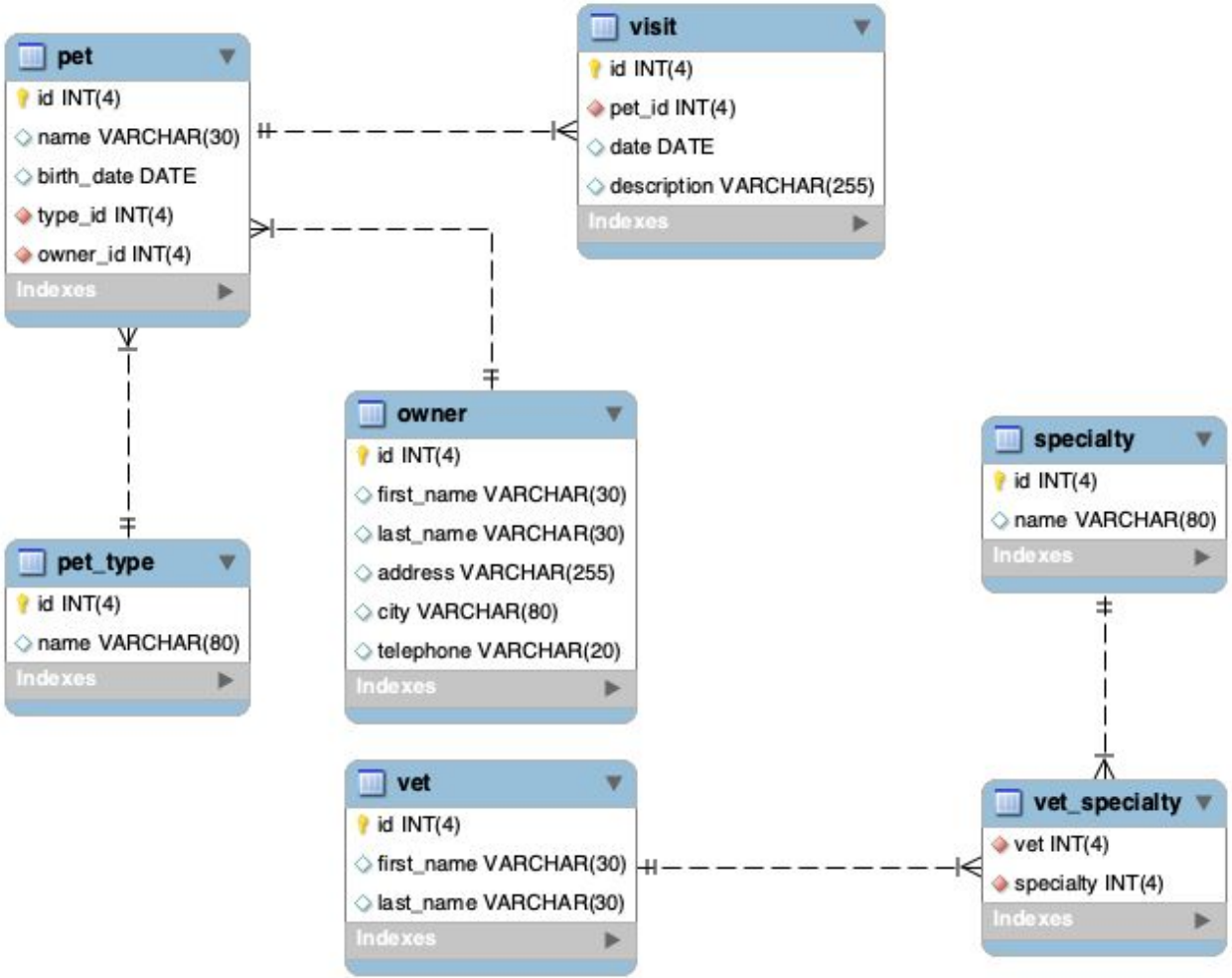
Use-Case:

A User opens the owner “ALL” page

Workflow:

List all owners

SQL

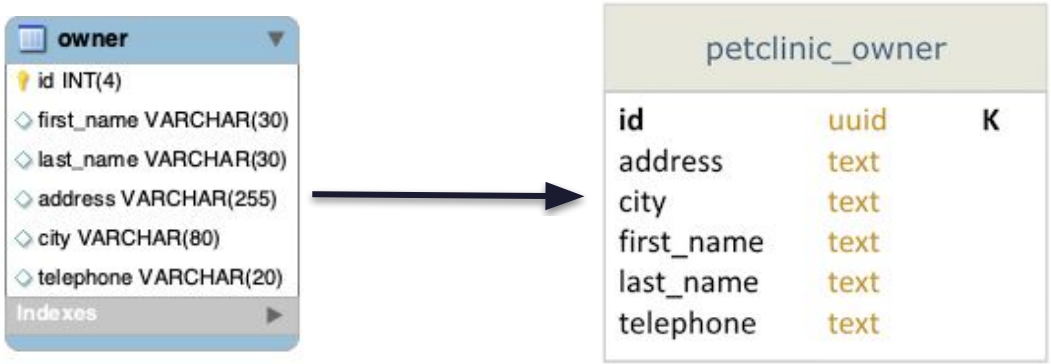

pet
id INT(4)
name VARCHAR(30)
birth_date DATE
type_id INT(4)
owner_id INT(4)
Indexes
visit
id INT(4)
pet_id INT(4)
date DATE
description VARCHAR(255)
Indexes
owner
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
address VARCHAR(255)
city VARCHAR(80)
telephone VARCHAR(20)
Indexes
pet_type
id INT(4)
name VARCHAR(80)
Indexes
specialty
id INT(4)
name VARCHAR(80)
Indexes
vet
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
Indexes
vet_specialty
vet INT(4)
specialty INT(4)
Indexes

Owner – SQL

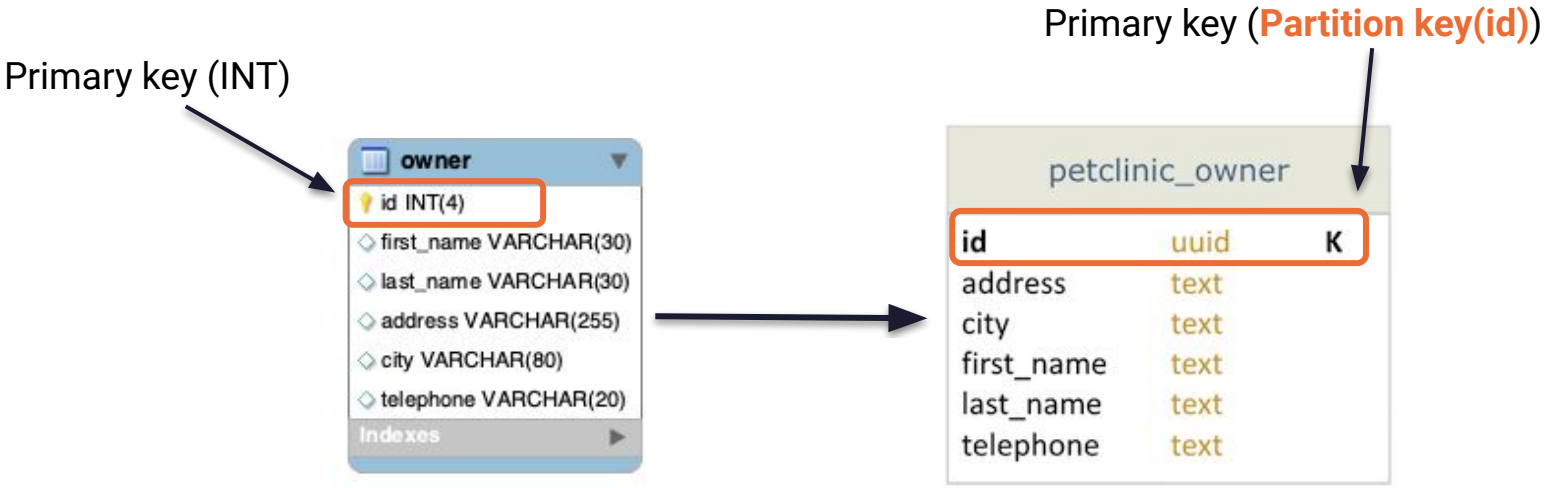
A simple list of owners and their properties

Primary key: **id**

Owner – SQL to CQL



Owner – SQL to CQL

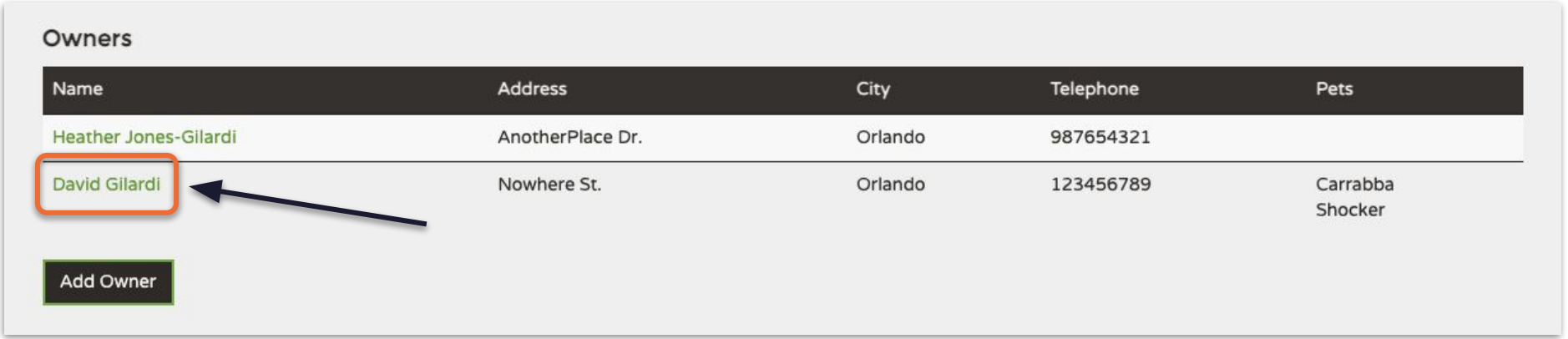


UUID's make good partition keys
Row is unique
One read gets ALL owner information

Owner Details

Use case 1: Owner to Pet

Owner to Pet – Application View



Owner to Pet – Application View

Owner Information

Name	David Gilardi
Address	Nowhere St.
City	Orlando
Telephone	123456789

< Back | Edit Owner | Add New Pet

Pets and Visits

Name	Shocker
Birth Date	2021/03/29
Type	dog

Edit Pet | Delete Pet | Add Visit

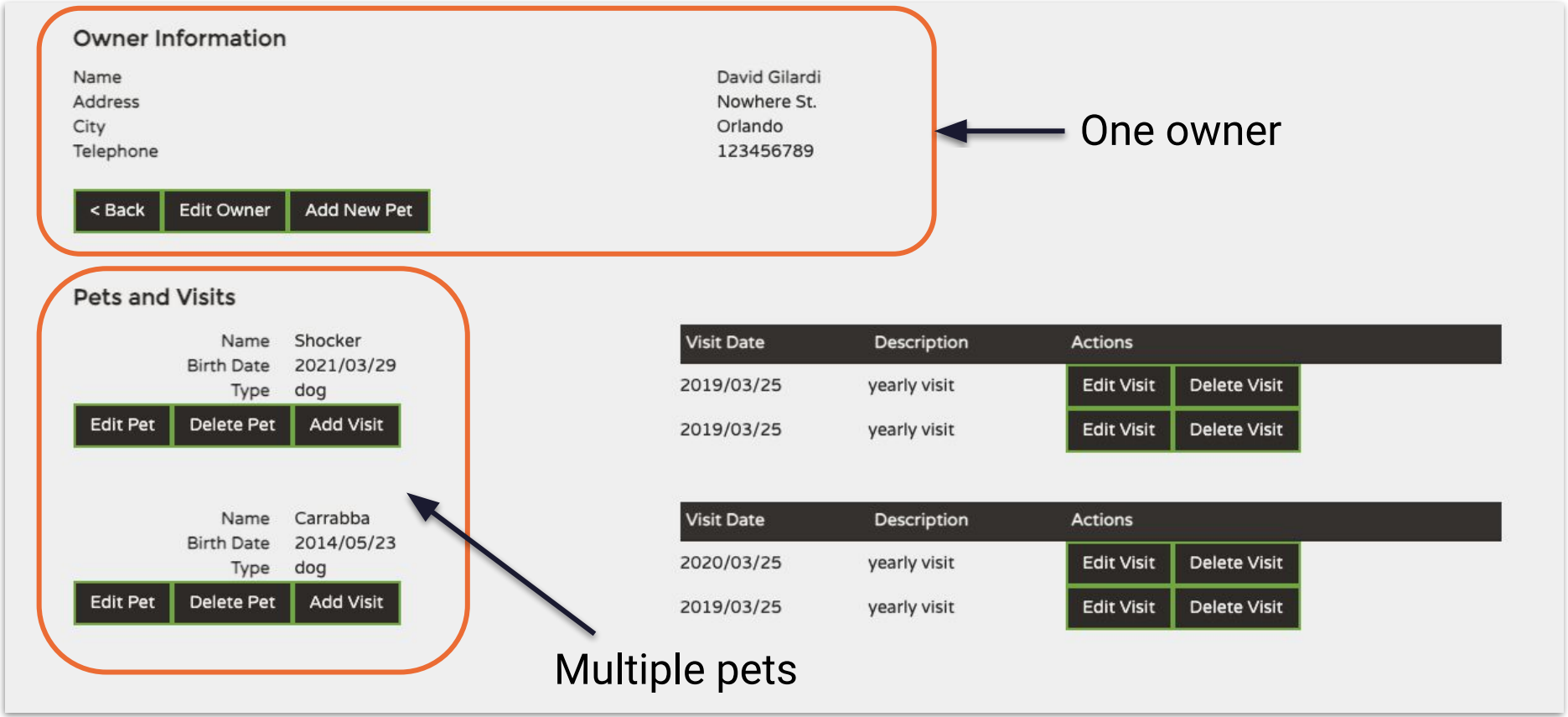
Visit Date	Description	Actions	
2019/03/25	yearly visit	Edit Visit	Delete Visit
2019/03/25	yearly visit	Edit Visit	Delete Visit

Name	Carrabba
Birth Date	2014/05/23
Type	dog

Edit Pet | Delete Pet | Add Visit

Visit Date	Description	Actions	
2020/03/25	yearly visit	Edit Visit	Delete Visit
2019/03/25	yearly visit	Edit Visit	Delete Visit

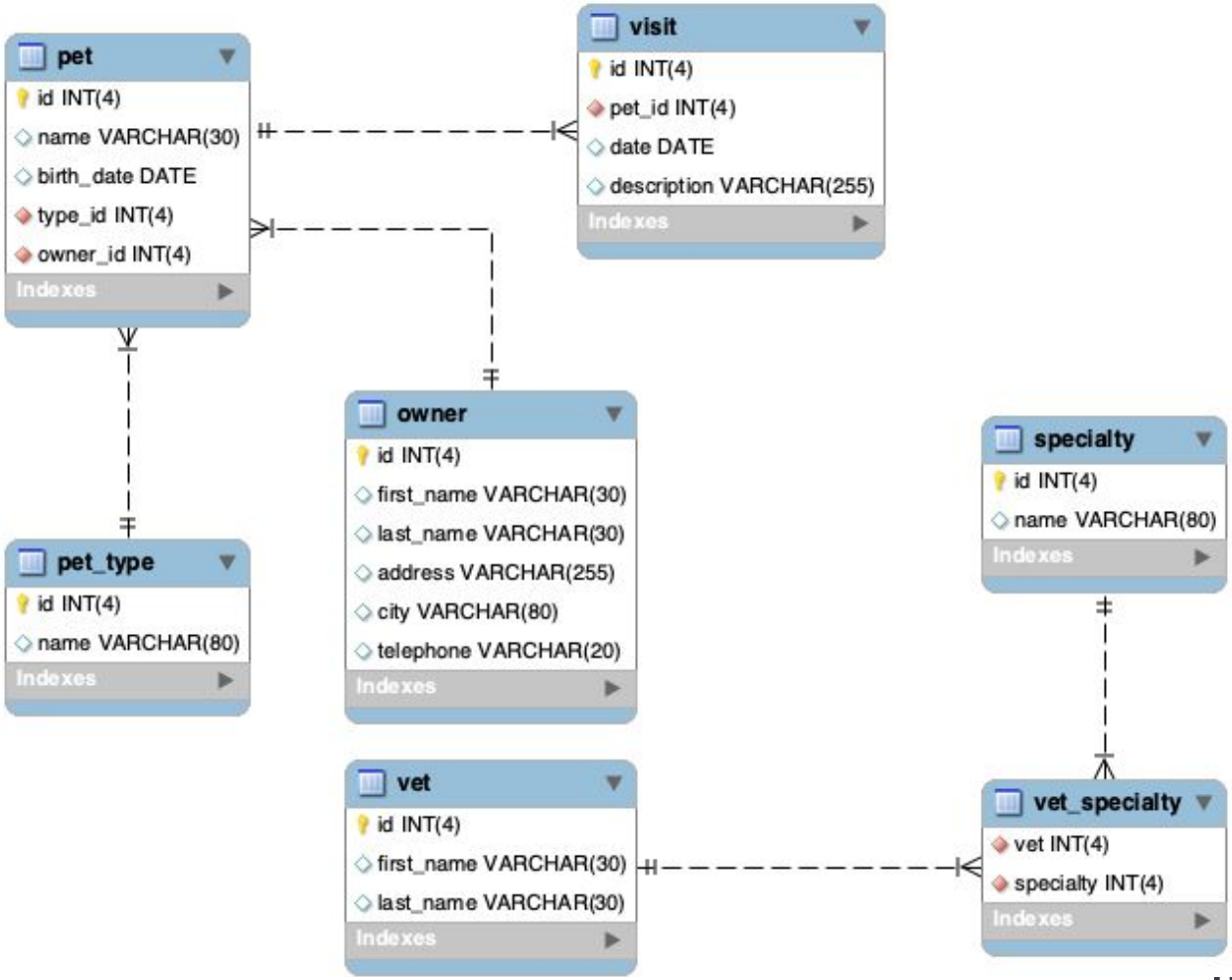
Owner to Pet – Application View



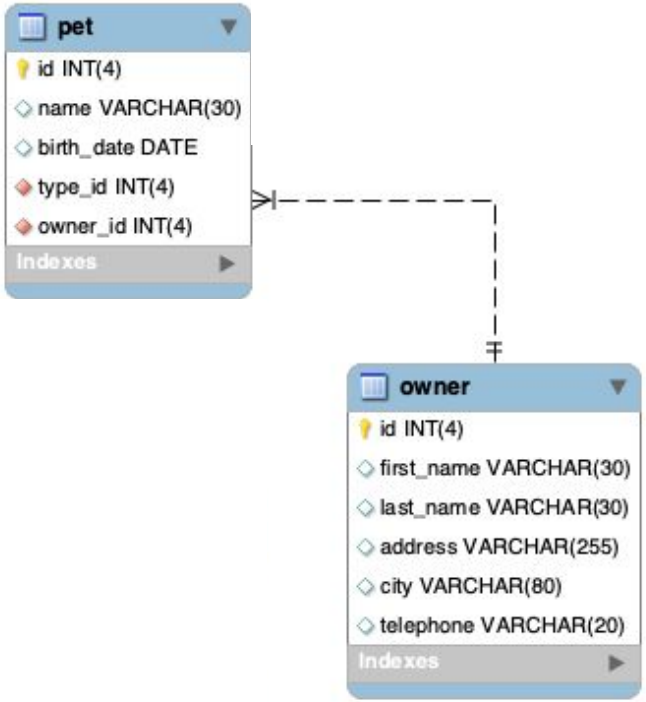
Use-Case:
A User opens the owner detail page

Workflow:
Find owner and any pets related to target owner using its identifier (owner_id)

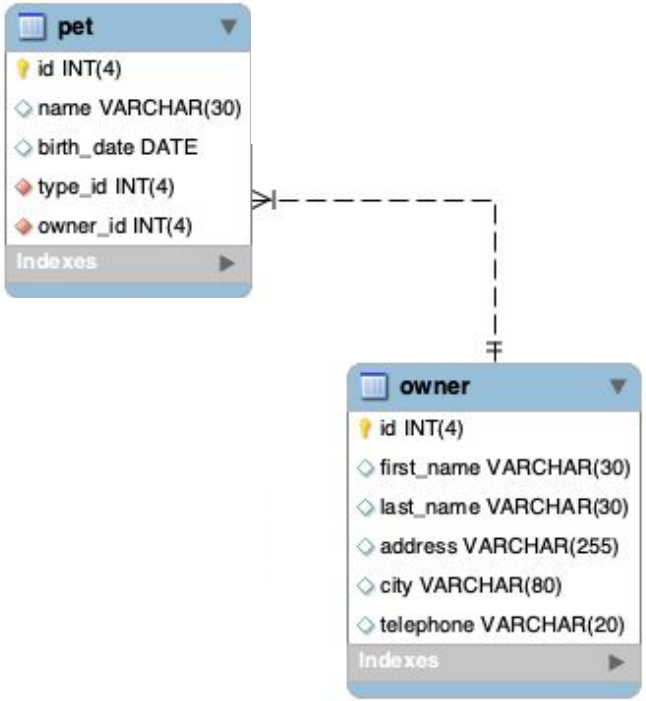
SQL


pet
id INT(4)
name VARCHAR(30)
birth_date DATE
type_id INT(4)
owner_id INT(4)
Indexes
visit
id INT(4)
pet_id INT(4)
date DATE
description VARCHAR(255)
Indexes
owner
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
address VARCHAR(255)
city VARCHAR(80)
telephone VARCHAR(20)
Indexes
pet_type
id INT(4)
name VARCHAR(80)
Indexes
specialty
id INT(4)
name VARCHAR(80)
Indexes
vet
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
Indexes
vet_specialty
vet INT(4)
specialty INT(4)
Indexes

Owner to Pet – SQL



Owner to Pet – SQL

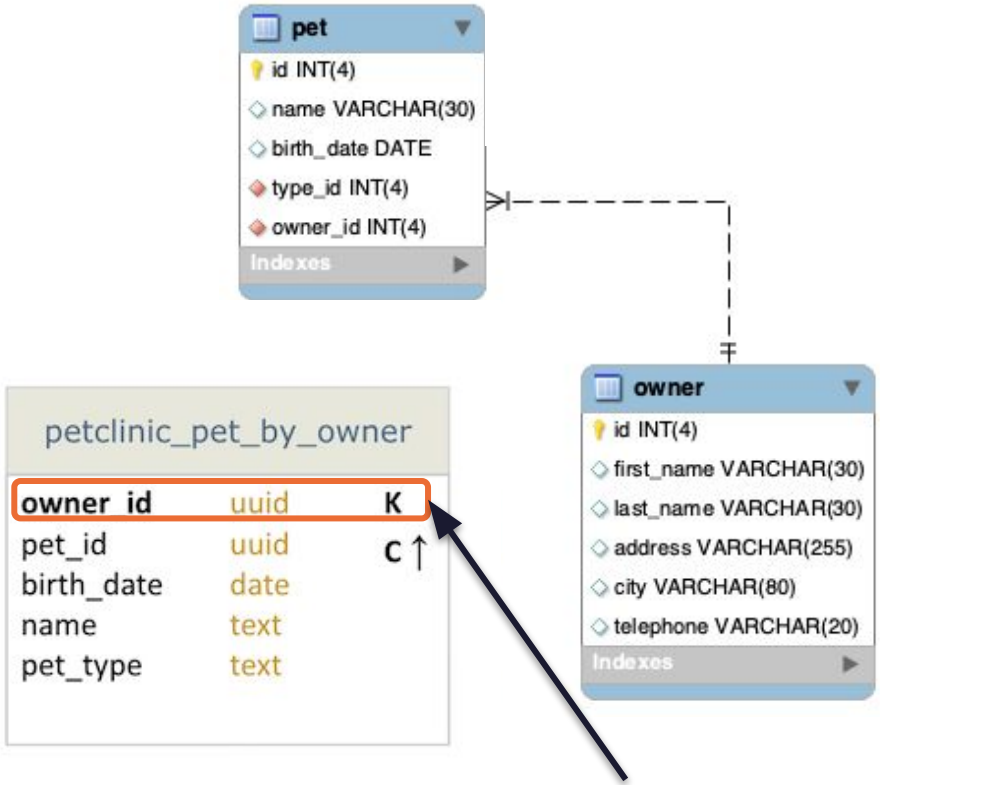


One to **Many** relationship

An owner can have multiple pets, but a pet can only have one owner

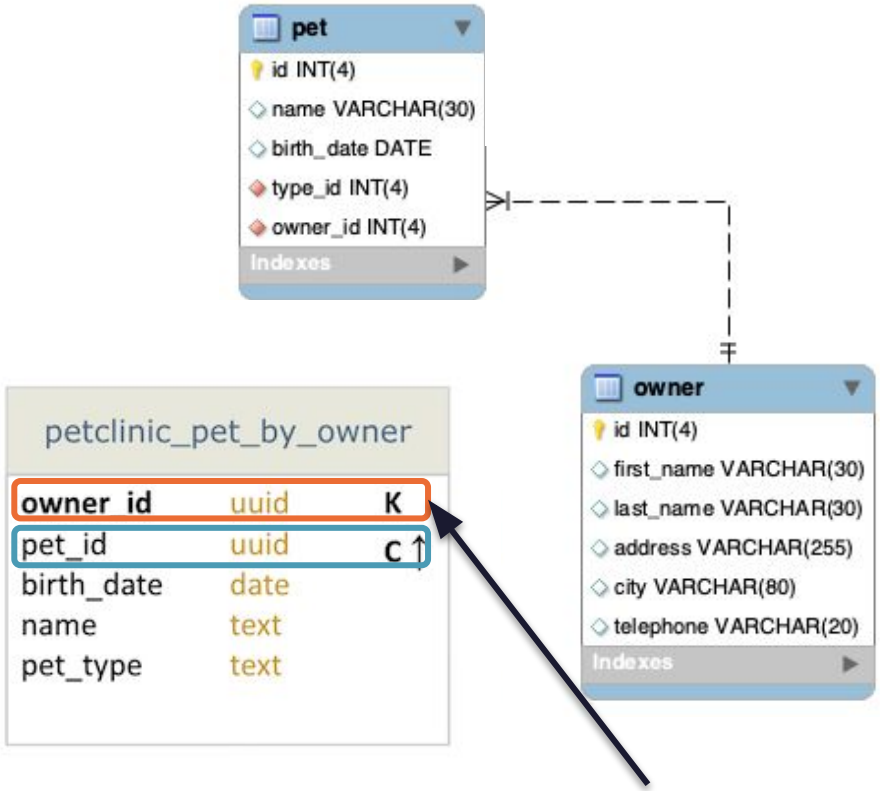
relationship: **id** to **owner_id (**foreign key)

Owner to Pet – SQL to CQL



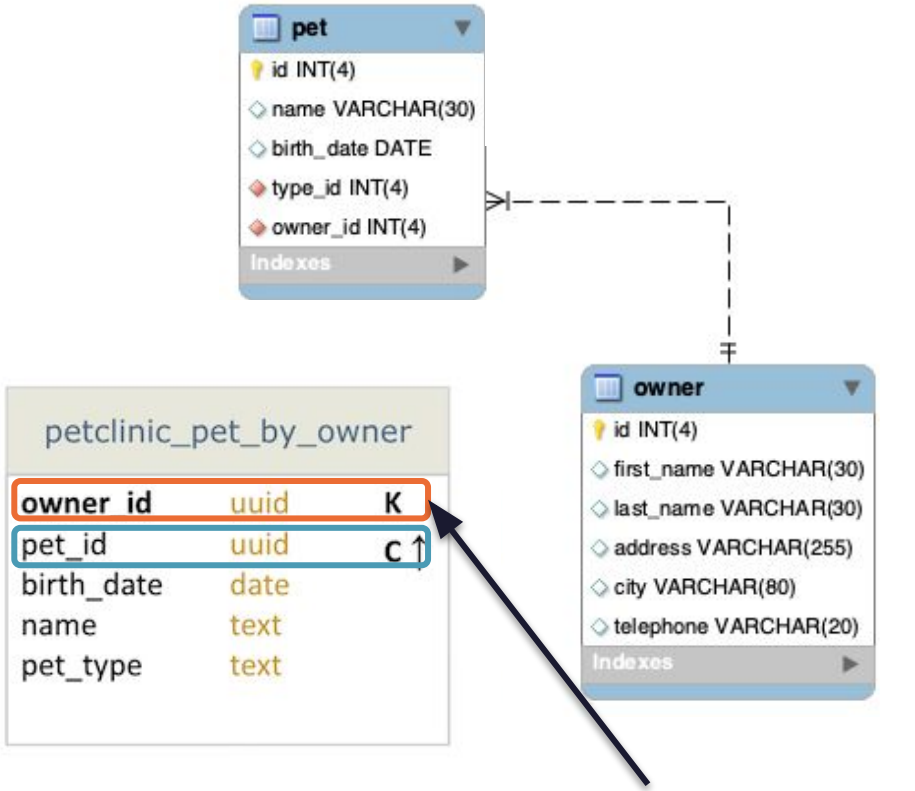
Primary key (**Partition key(owner_id)**)

Owner to Pet – SQL to CQL



Primary key (
Partition key(owner_id)
Clustering column(pet_id))

Owner to Pet – SQL to CQL



Primary key (
Partition key(owner_id)
Clustering column(pet_id))

When using **multiple attributes** (birth_date, name, pet_type)

Using a clustering column = **multi-row partition**

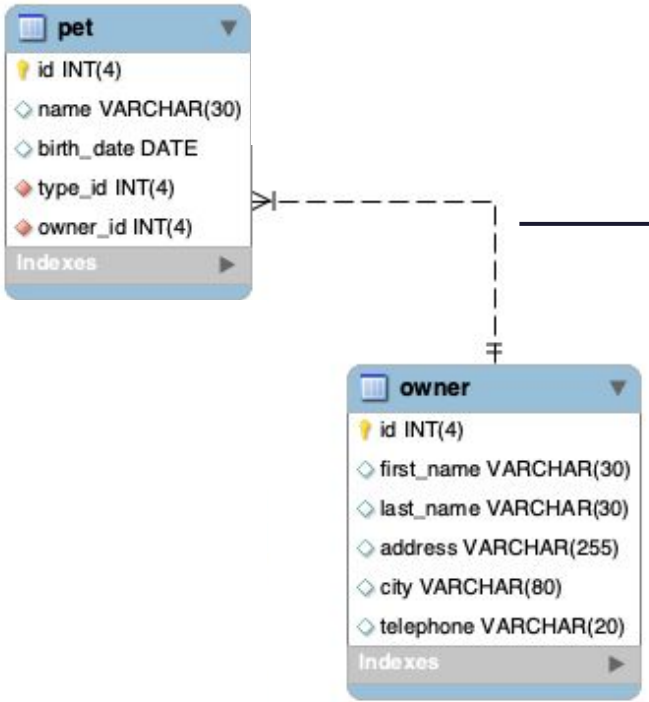
Means for every owner partition there can be **1 to n rows of pets** on a single read

One to **Many**

What's another option?

Another option

- Make owner fields STATIC

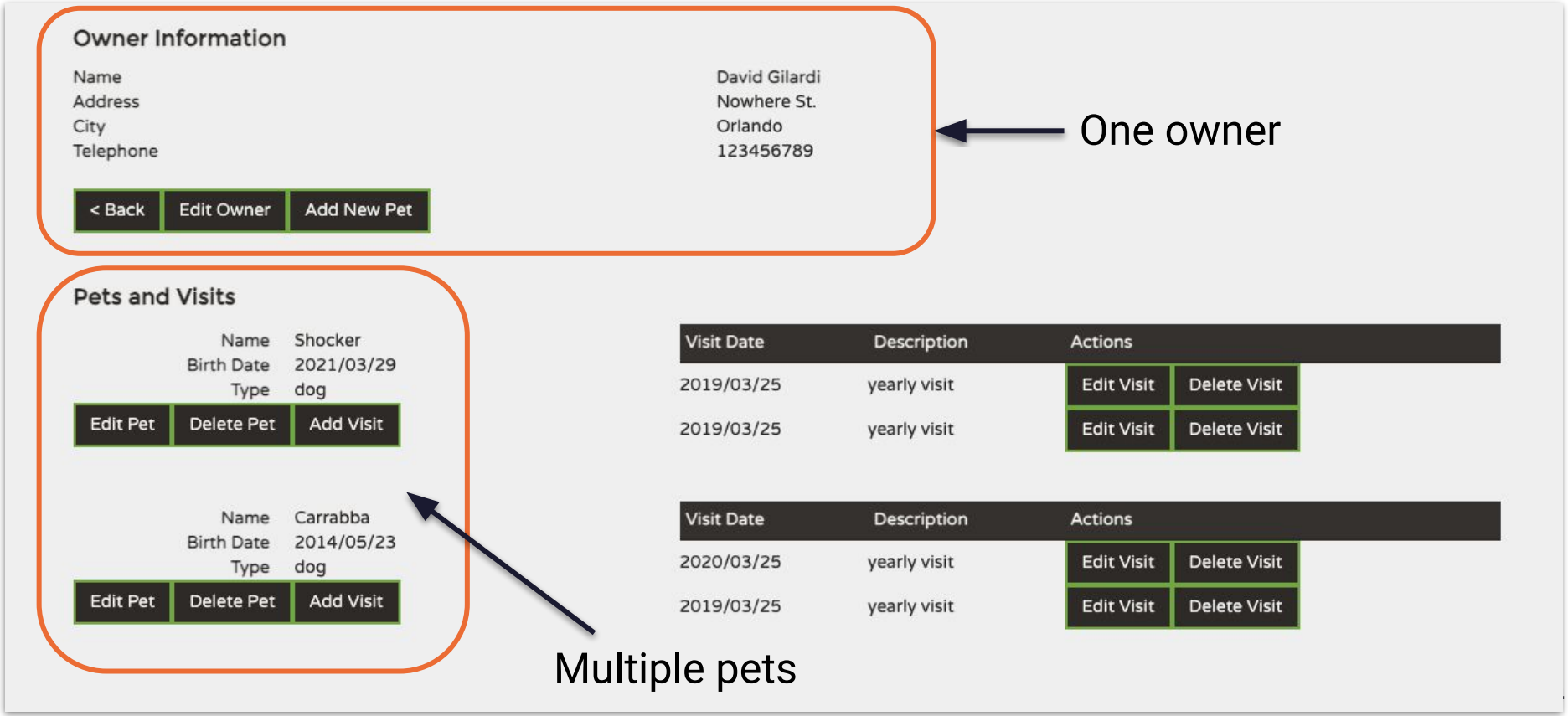


petclinic_pet_by_owner		
owner_id	uuid	K
pet_id	uuid	C ↑
birth_date	date	
name	text	
pet_type	text	
address	text	S
city	text	S
first_name	text	S
last_name	text	S
telephone	text	S

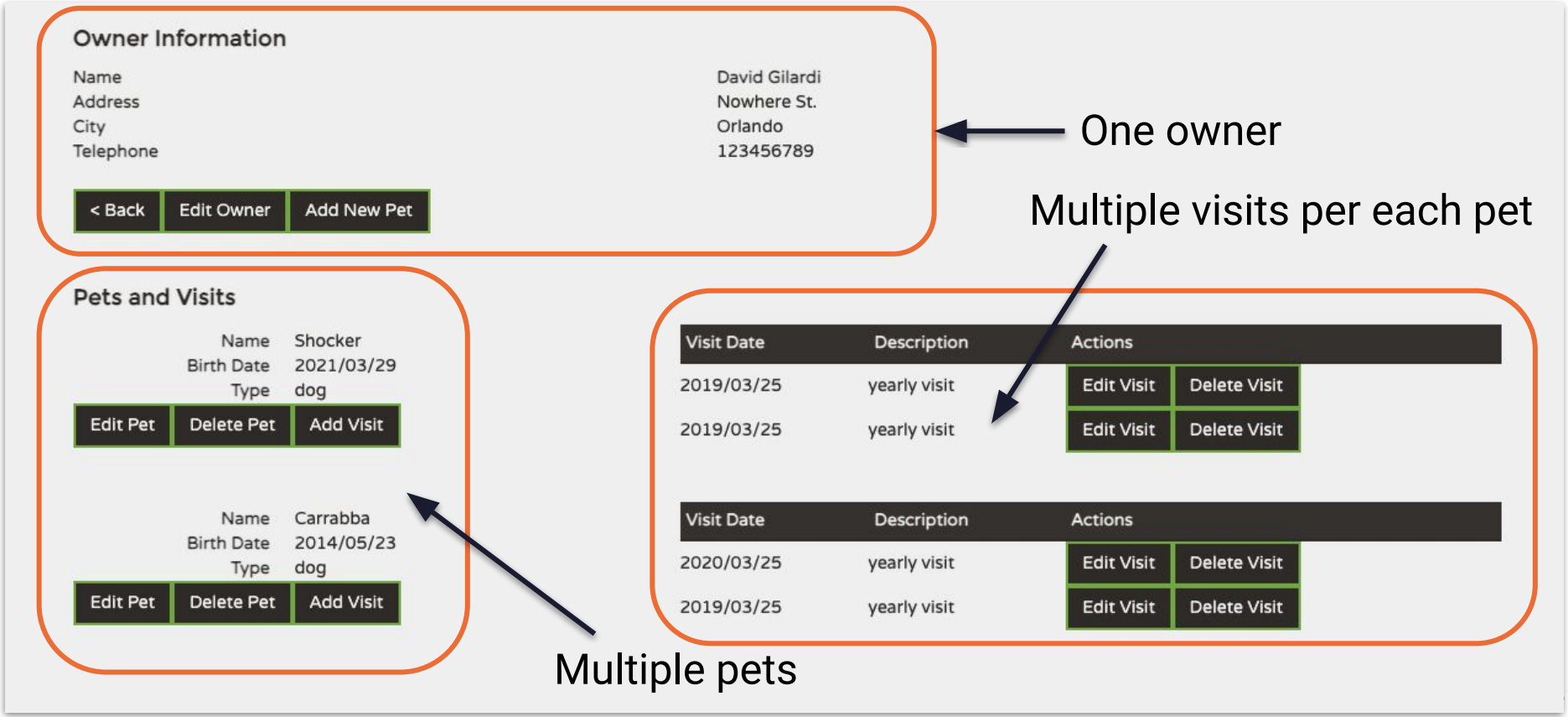
Owner Details

Use case 2: Pet to Visit

Owner to Pet – Application View



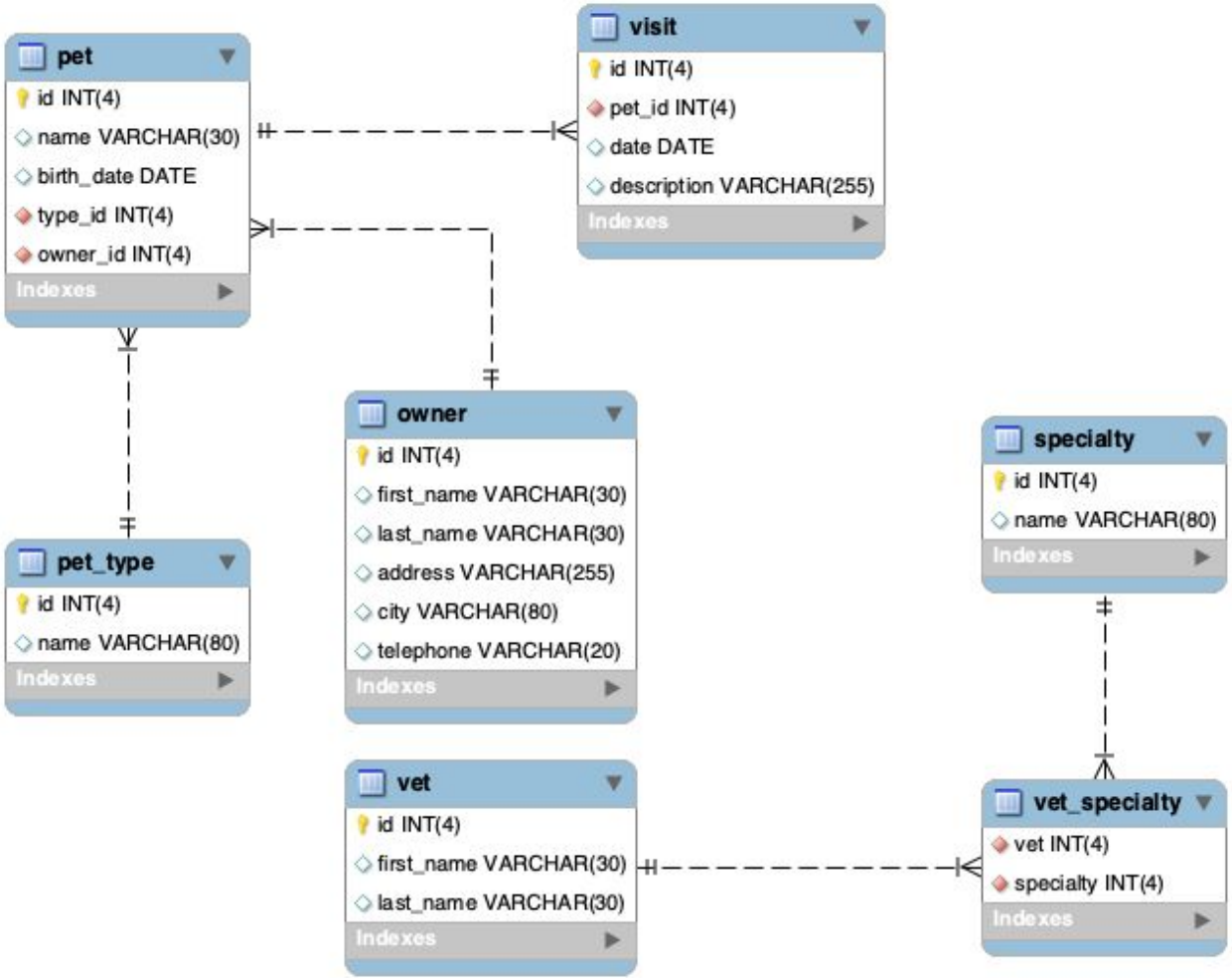
Owner to Pet – Application View

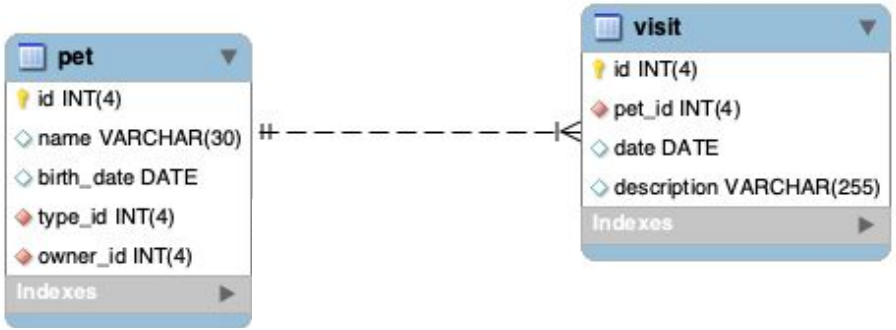


Use-Case:
A User opens the owner detail page

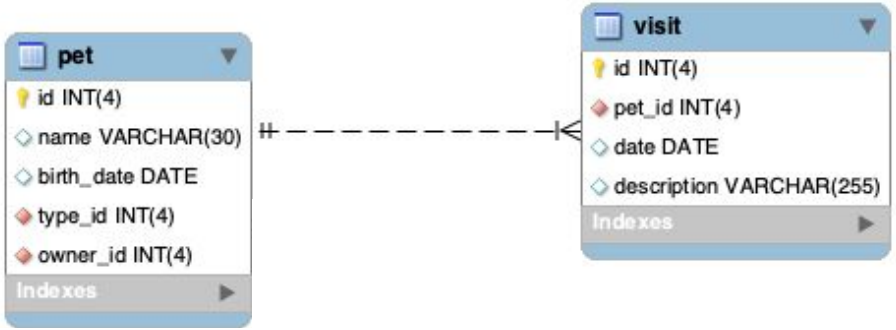
Workflow:
Find pets and any visits related to target pet using its identifier (pet_id)

SQL


pet
id INT(4)
name VARCHAR(30)
birth_date DATE
type_id INT(4)
owner_id INT(4)
Indexes
visit
id INT(4)
pet_id INT(4)
date DATE
description VARCHAR(255)
Indexes
owner
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
address VARCHAR(255)
city VARCHAR(80)
telephone VARCHAR(20)
Indexes
pet_type
id INT(4)
name VARCHAR(80)
Indexes
specialty
id INT(4)
name VARCHAR(80)
Indexes
vet
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
Indexes
vet_specialty
vet INT(4)
specialty INT(4)
Indexes



Pet to Visit – SQL



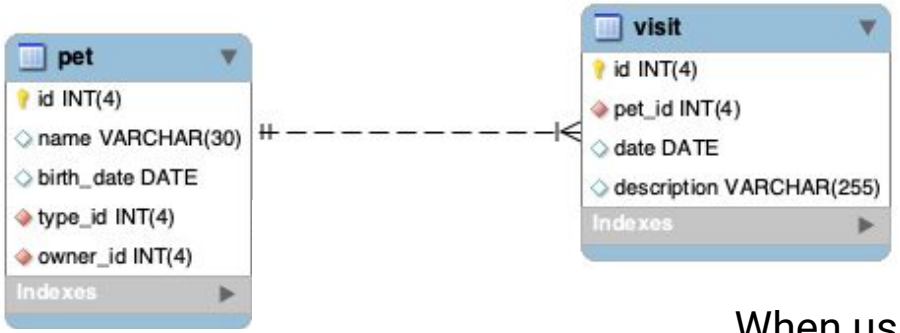
Pet to Visit – SQL

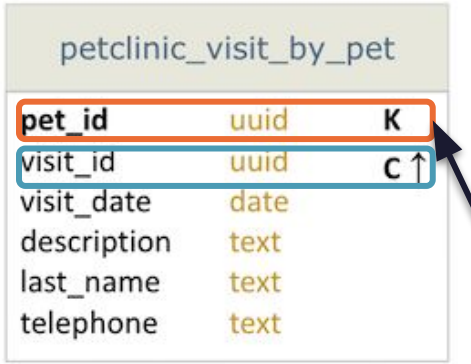
One to **Many** relationship

A pet can have multiple visits, but a visit can only have one pet

relationship: **id** to **pet_id** (foreign key)

Pet to Visit – SQL





When using **multiple attributes** (visit_date, description, last_name, telephone)

Using a clustering column = **multi-row partition**

Means for every pet partition there can be **1 to n rows of visits** on a single read

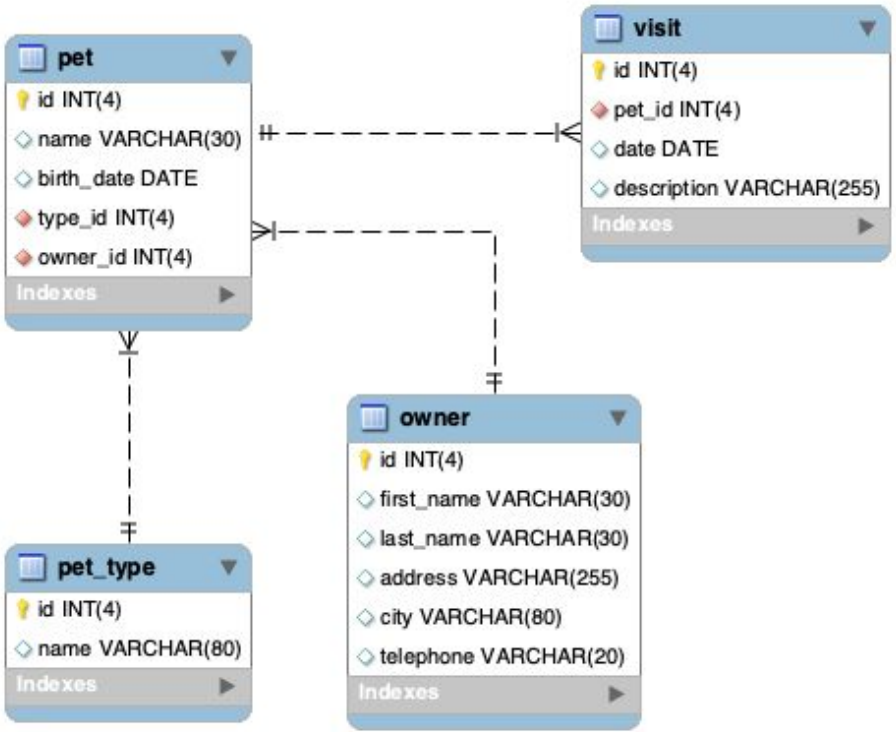
One to **Many**

Primary key (
Partition key(pet_id)
Clustering column(visit_id))

What's another option?

Another option

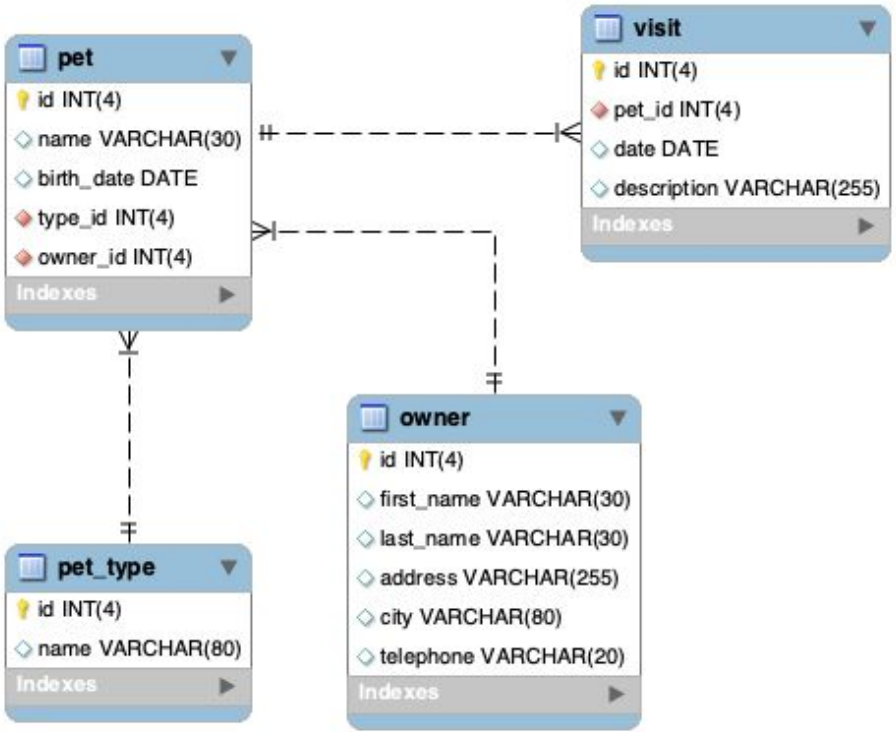
- Denormalize even more



petclinic_pet_by_owner		
owner_id	uuid	K
pet_id	uuid	C ↑
visit_id	uuid	C ↑
pet_birth_date	date	
pet_name	text	
pet_type	text	
owner_address	text	S
owner_city	text	S
owner_first_name	text	S
owner_last_name	text	S
owner_telephone	text	S
visit_date	date	
visit_description	text	

Another option

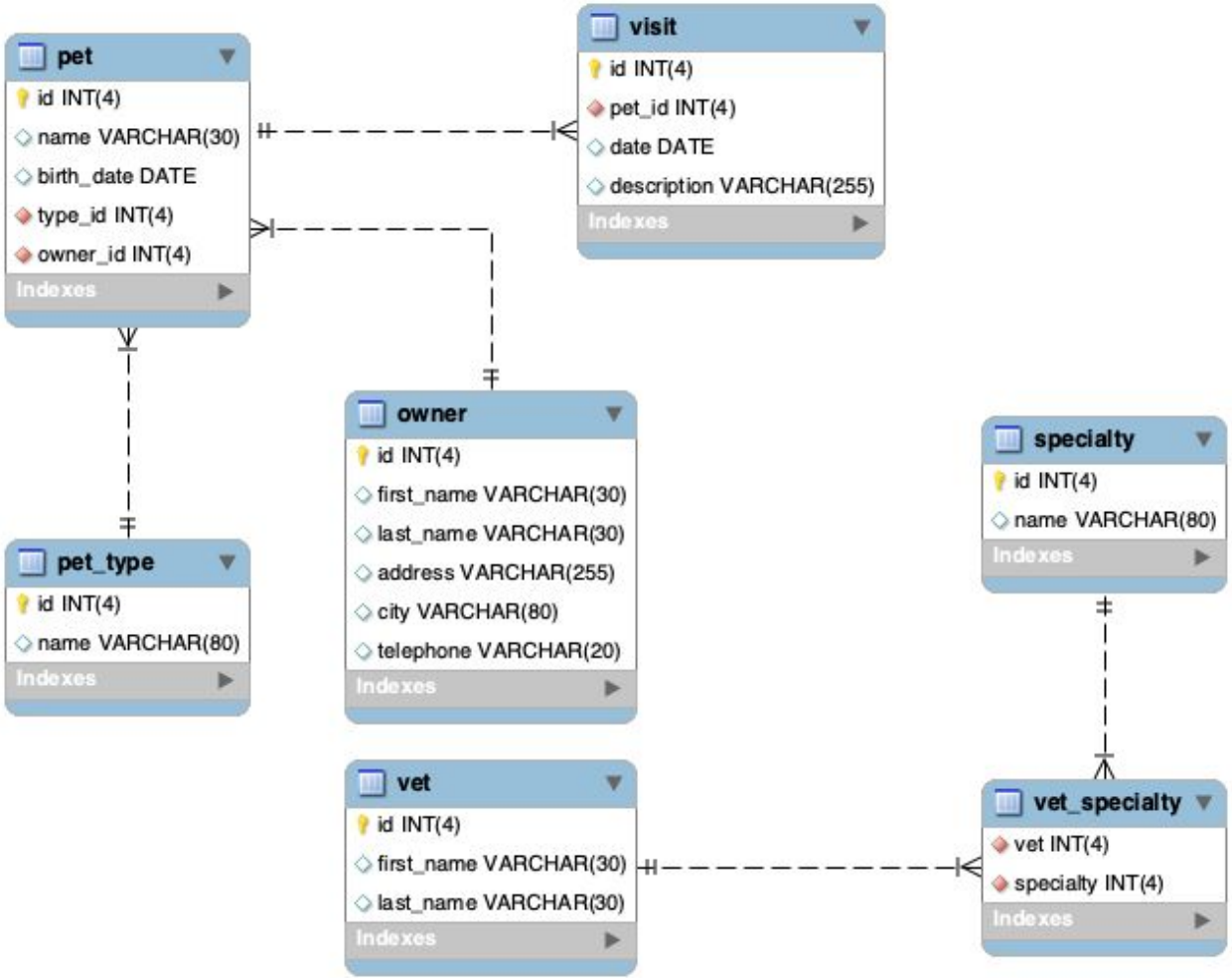
- Use a document




```
{
  "data":{
     "Joey":{
        "firstname":"Joey",
        "lastname":"Doe",
        "weights":{
               "reps":15,
               "type":"bench press",
               "weight":150
        }
     },
     "Janet":{
        "email":"janet.doe@gmail.com",
        "favorite color":"grey",
        "firstname":"Janet",
        "lastname":"Doe"
     },
     "3ffc7ae6-c42d-46de-b52b-b5e77ae6a87b":{
        "id":"some-stuff",
        "other":"This is nonsensical stuff."
     }
  }
}
```

Final models

SQL to NoSQL


pet
id INT(4)
name VARCHAR(30)
birth_date DATE
type_id INT(4)
owner_id INT(4)
Indexes
visit
id INT(4)
pet_id INT(4)
date DATE
description VARCHAR(255)
Indexes
pet_type
id INT(4)
name VARCHAR(80)
Indexes
owner
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
address VARCHAR(255)
city VARCHAR(80)
telephone VARCHAR(20)
Indexes
specialty
id INT(4)
name VARCHAR(80)
Indexes
vet
id INT(4)
first_name VARCHAR(30)
last_name VARCHAR(30)
Indexes
vet_specialty
vet INT(4)
specialty INT(4)
Indexes


petclinic_owner
id uuid K
address text
city text
first_name text
last_name text
telephone text
petclinic_pet_by_owner
owner_id uuid K
pet_id uuid C↑
birth_date date
name text
pet_type text
petclinic_visit_by_pet
pet_id uuid K
visit_id uuid C↑
visit_date date
description text
last_name text
telephone text
petclinic_vet
id uuid K
first_name text
last_name text
specialties set<text>
petclinic_vet_by_specialty
specialty text K
vet_id uuid C↑
first_name text
last_name text
petclinic_reference_lists
list_name text K
values set<text>

HandsOn #2 Create Data Model

DataStax Astra

Get your instance here:

- dtsx.io/workshop

GitHub

Repository:

- dtsx.io/sql-to-nosql

DataStax Developers

Agenda

01

Why switch from SQL to NoSQL?

02

Three migration approaches

03

Data modeling SQL vs NoSQL

04

Migrate your data model

05

Moving data with DSBulk

06

Resources

Moving data with DSBulk

HandsOn #3 and #4

DataStax Astra

Get your instance here:

- dtsx.io/workshop

GitHub

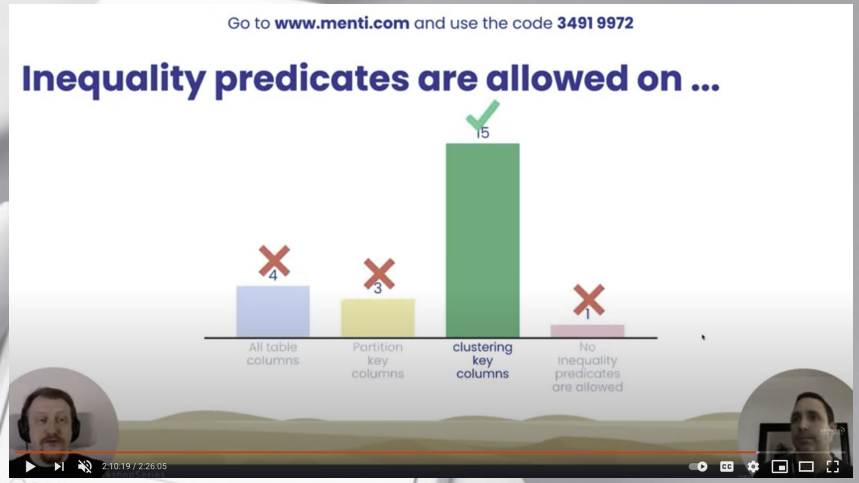
Repository:

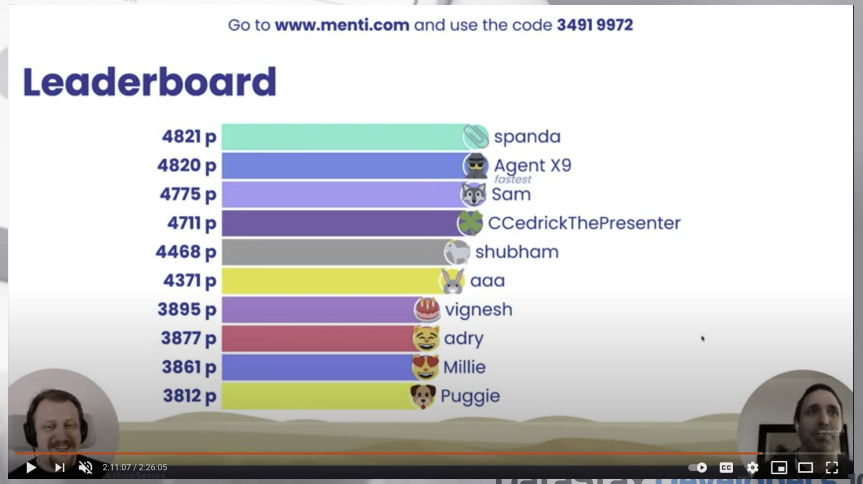
- dtsx.io/sql-to-nosql

DataStax Developers

menti.com


Apache
CASSANDRA
25th
DS COMICS
DATASTAX


Go to www.menti.com and use the code 3491 9972
Inequality predicates are allowed on ...
4
3
15
1
All table columns
Partition key columns
clustering key columns
No inequality predicates are allowed
2:10:19 / 2:26:05


Go to www.menti.com and use the code 3491 9972
Leaderboard
4821 p spanda
4820 p Agent X9
fastest
4775 p Sam
4711 p CCedrickThePresenter
4468 p shubham
4371 p aaa
3895 p vignesh
3877 p adry
3861 p Millie
3812 p Puggie
2:11:07 / 2:26:05

Agenda

01

Why switch from SQL to NoSQL?

02

Three migration approaches

03

Data modeling SQL vs NoSQL

04

Migrate your data model

05

Moving data with DSBulk

06

Resources

Awesome-cassandra - Cassandra from Relational

- Multiple blogs
  - https://github.com/anant/awesome-cassandra#cassandra-from-relational

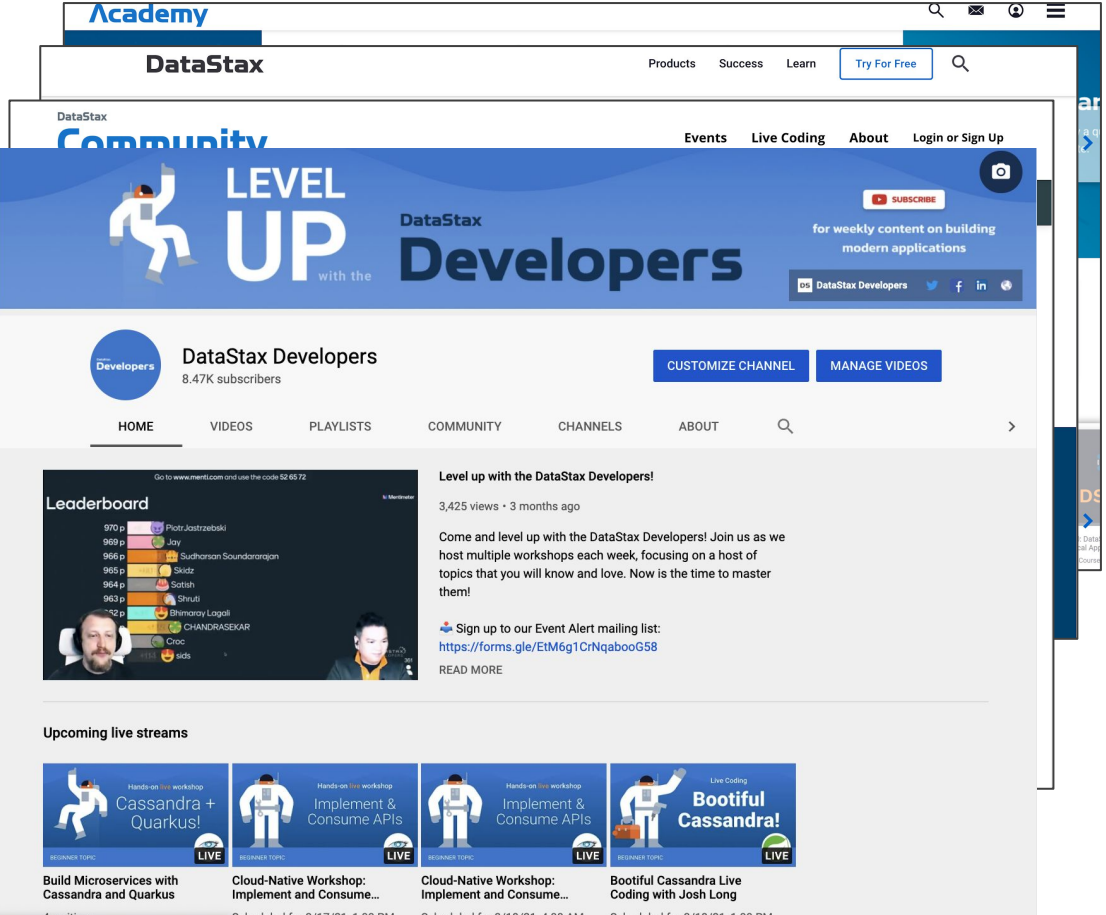
Best Practices for Migrating from a Relational Data Platform to Apache Cassandra™

- Blog
  - https://www.datastax.com/blog/best-practices-migrating-relational-data-platform-apache-cassandratm
- Free on-demand webinar
  - https://www.datastax.com/resources/webinar/best-practices-migrating-relational-data-platform-apache-cassandratm

Migrating RDBMS Monoliths Into a Transformational Data Architecture

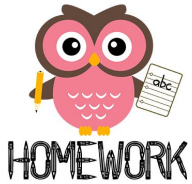
- Free on-demand webinar
  - https://www.datastax.com/resources/webinar/migrating-rdbms-monoliths-transformational-data-architecture

Developers



- ✔ Academy.datastax.com
- ✔ datastax.com/dev
- ✔ community.datastax.com
- ✔ Datastax Developers YouTube Channel

Homework (dtsx.io/sql-to-nosql)



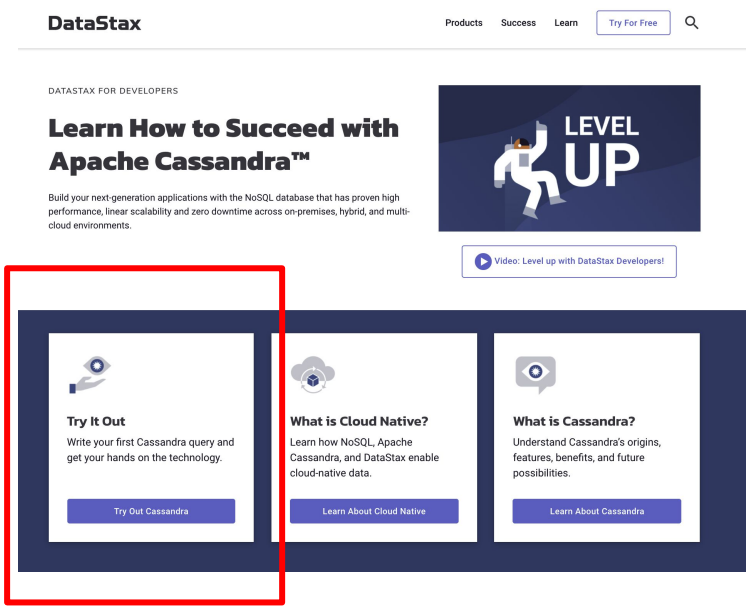
[Complete hands-on through to #4]

- Use DSbulk to load your SQL data into your NoSQL Cassandra data model

[Try other content] –

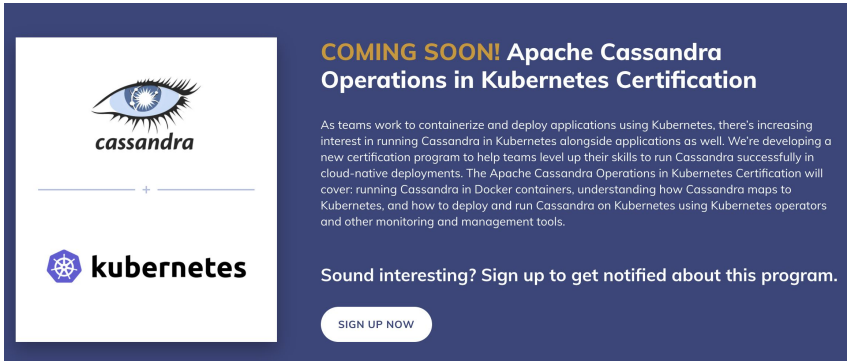
https://www.datastax.com/dev/scenario/try-it-out-cassandra-data-modeling

- Go to datastax/dev and use the try-it-out



Certifications

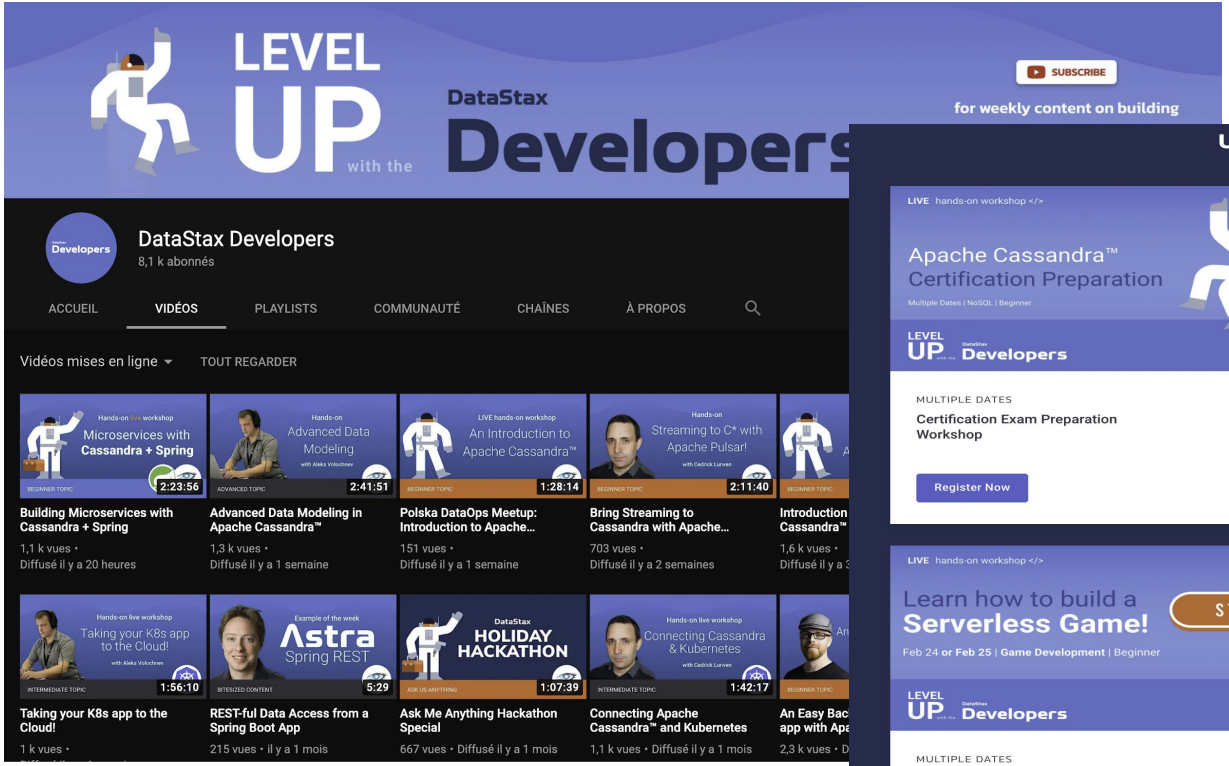
https://www.datastax.com/dev/certifications

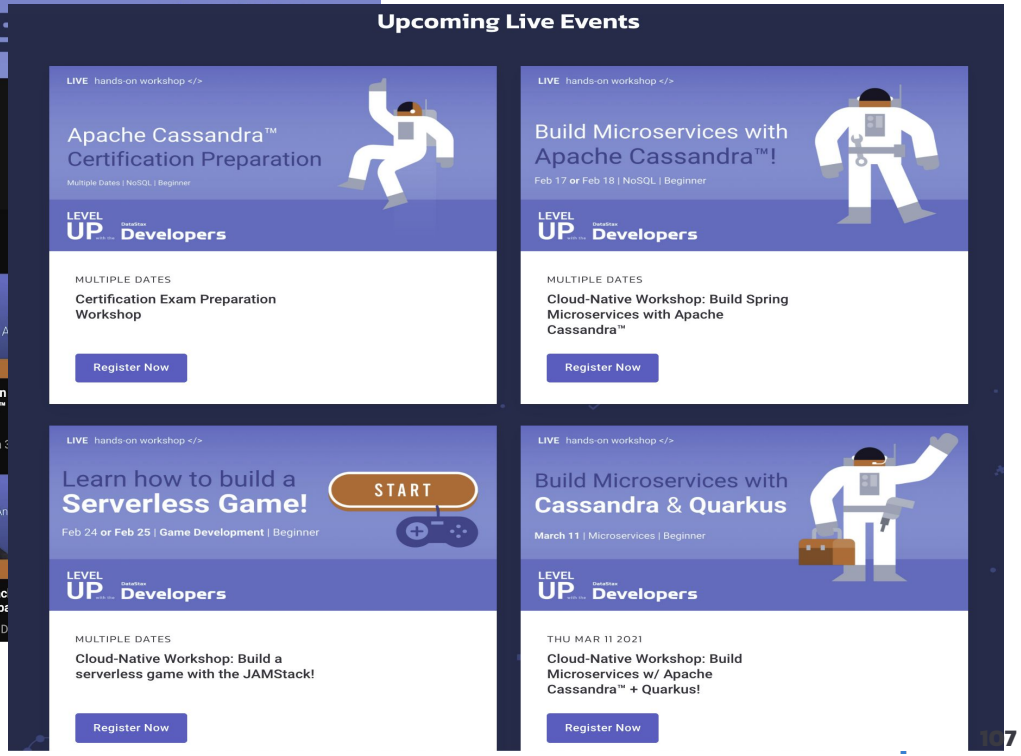


Vouchers (145$ each) , valid 3 months, with 2 attempts will be given to people who apply and register to the 3 episodes.

Weekly Workshops

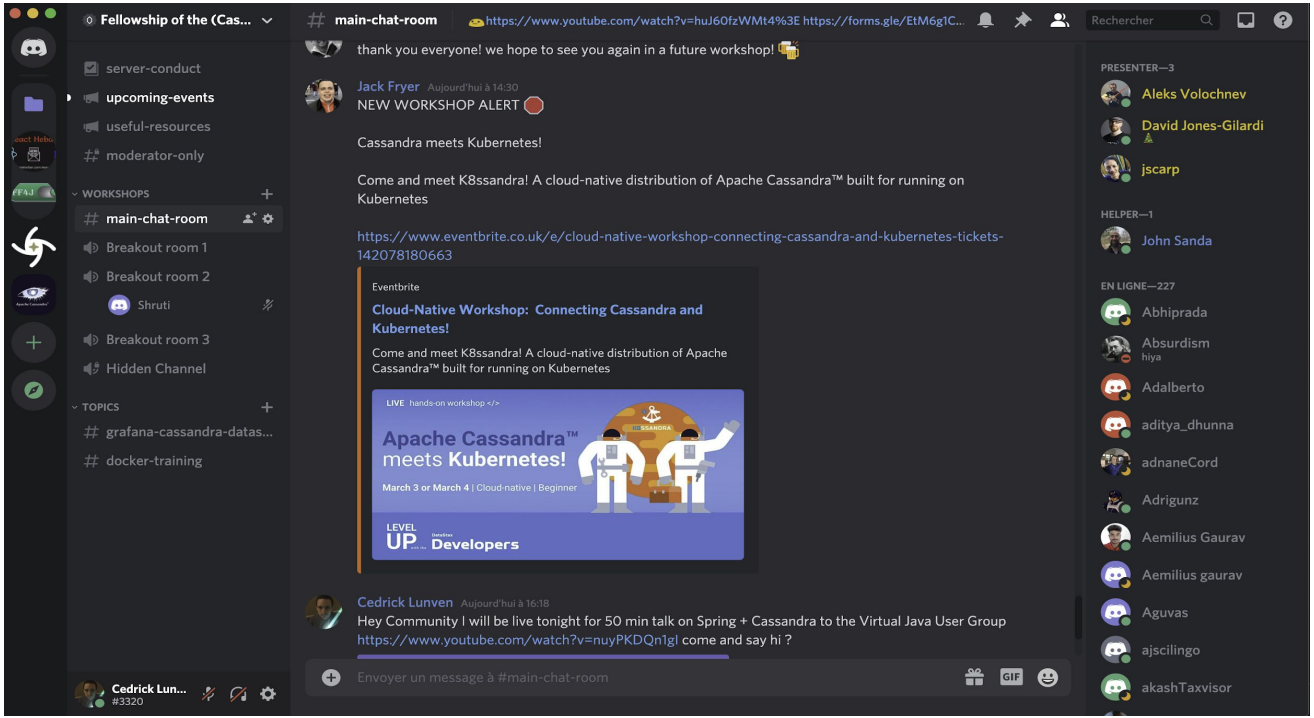
https://www.datastax.com/workshops



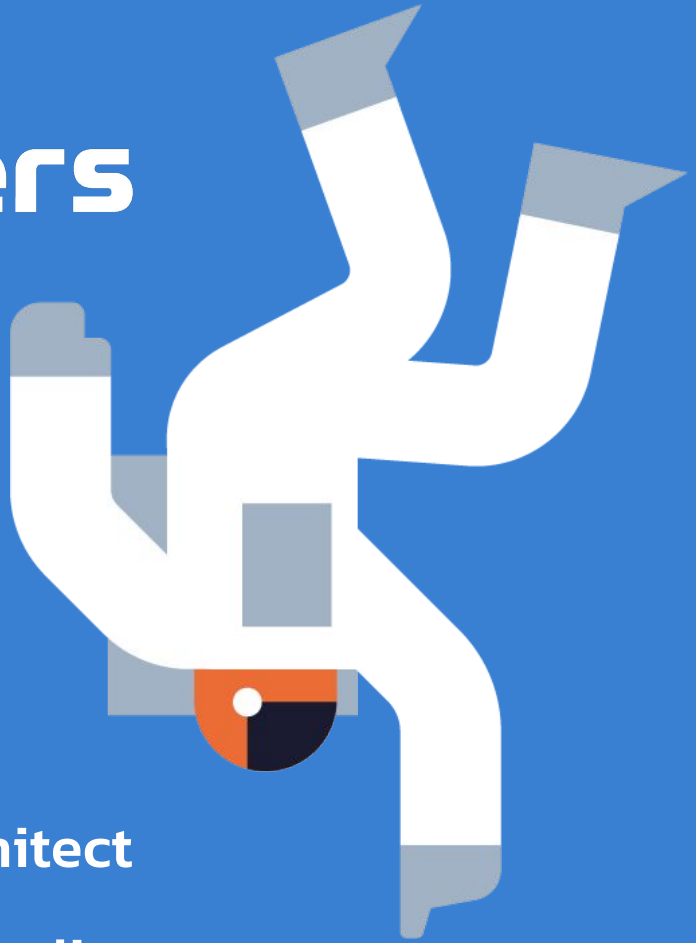


Join our 10k Discord Community
The Fellowship of the RINGS

https://bit.ly/cassandra-workshop



DataStax Developers



Thank you!

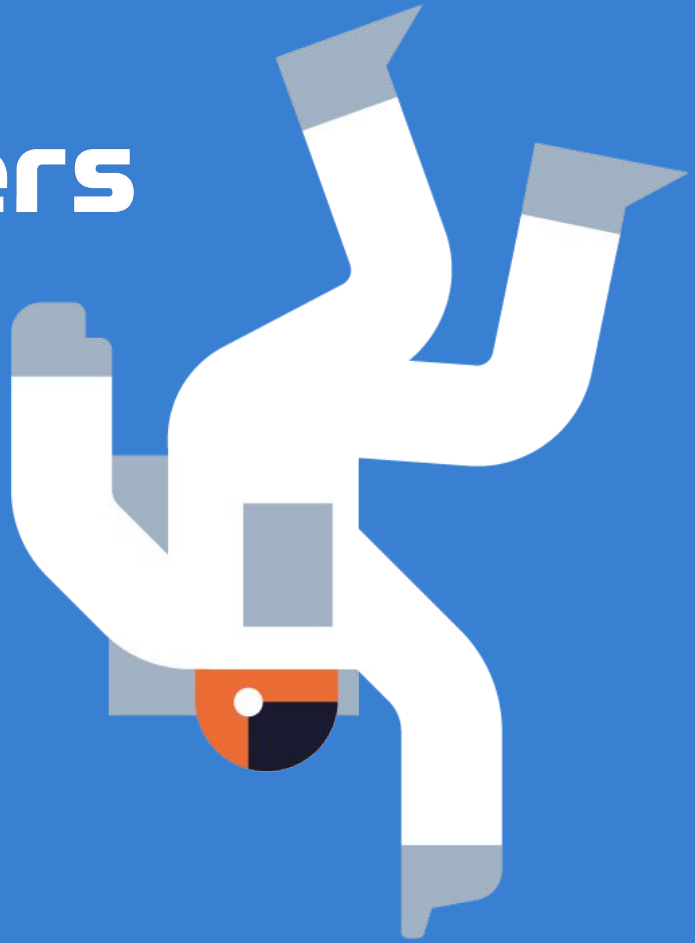
GitHub
@hadesarchitect
@clun
@SonicDMG

@hadesarchitect
@clunven
@SonicDMG

@hadesarchitect
@clunven
@david-gilardi

DataStax Developers

Thank you!

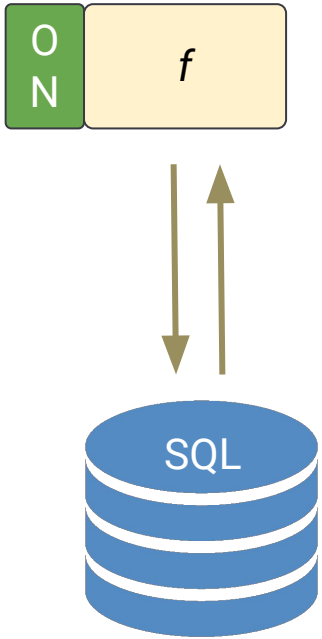


Subscribe

Approach 2: Zero-Downtime Migration with CDC

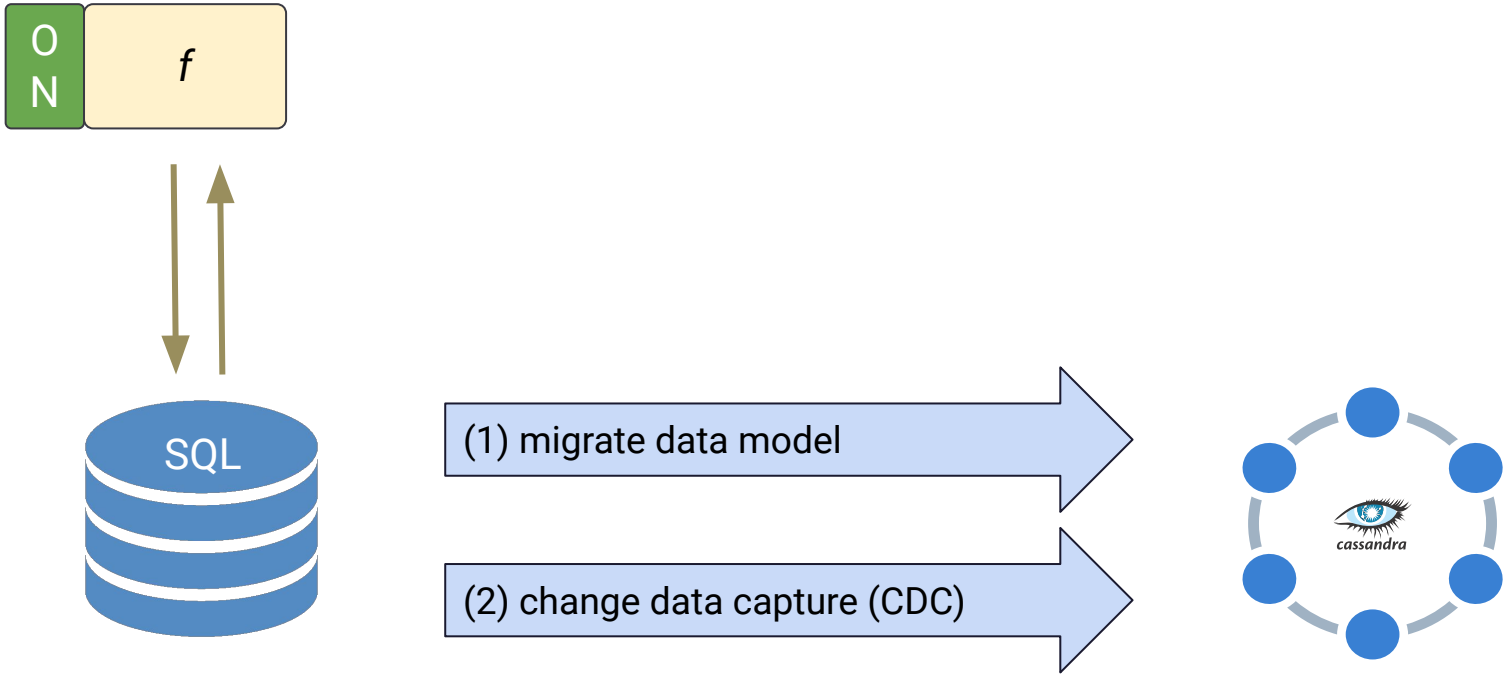
Zero-Downtime Migration with CDC

Approach 2



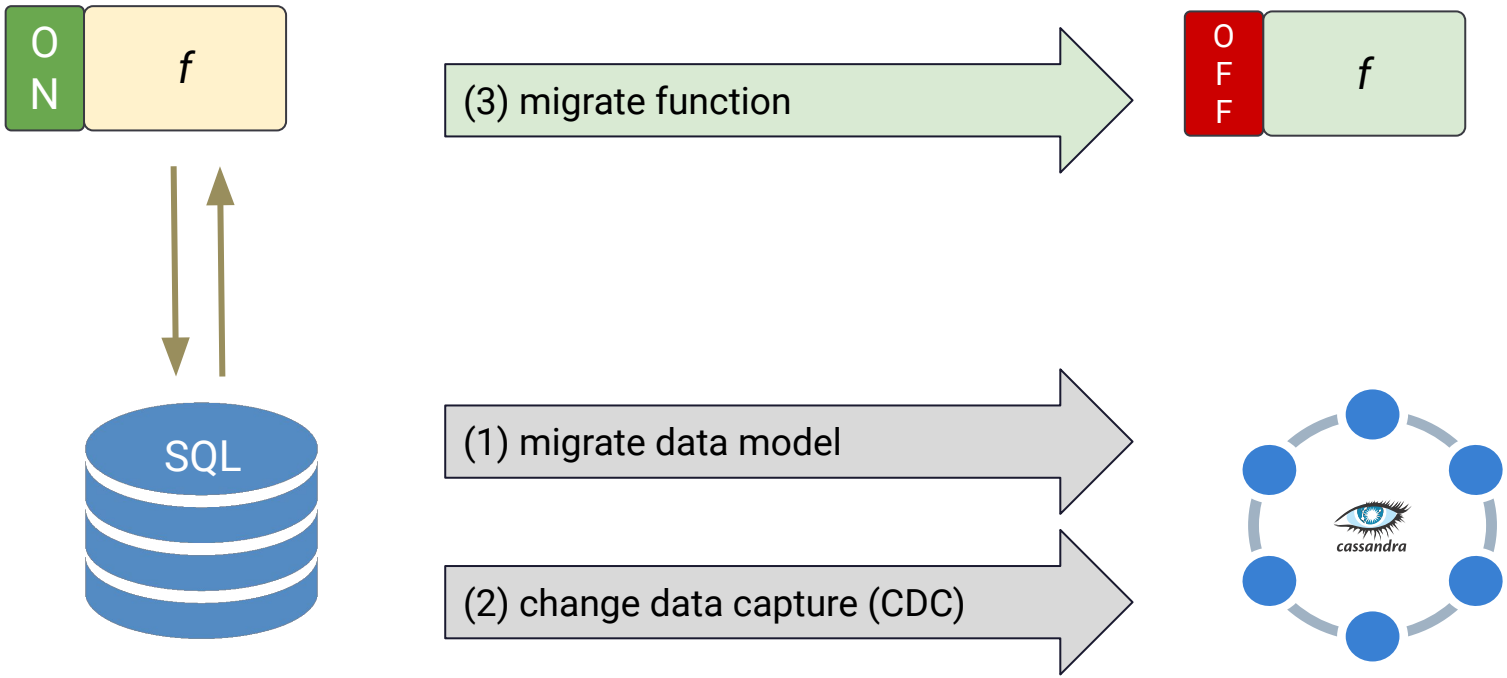
Zero-Downtime Migration with CDC

Approach 2



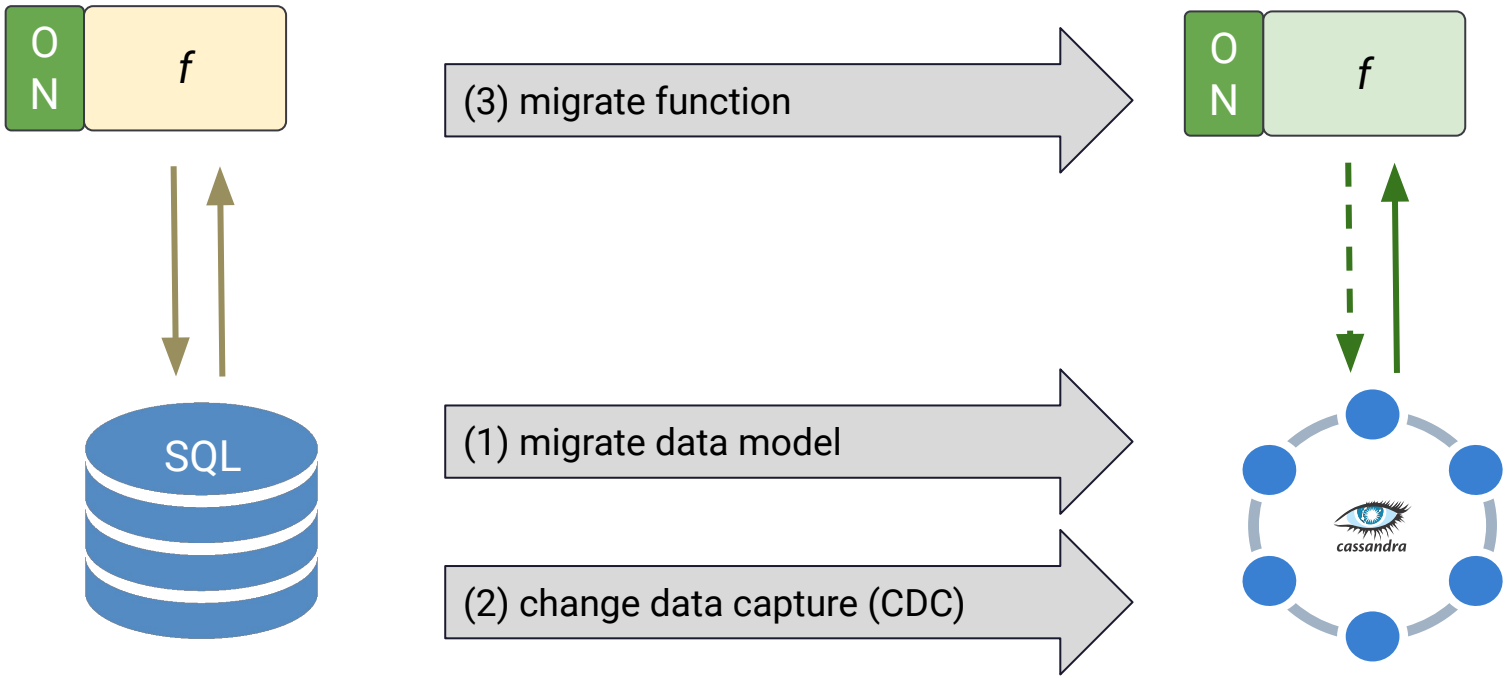
Zero-Downtime Migration with CDC

Approach 2



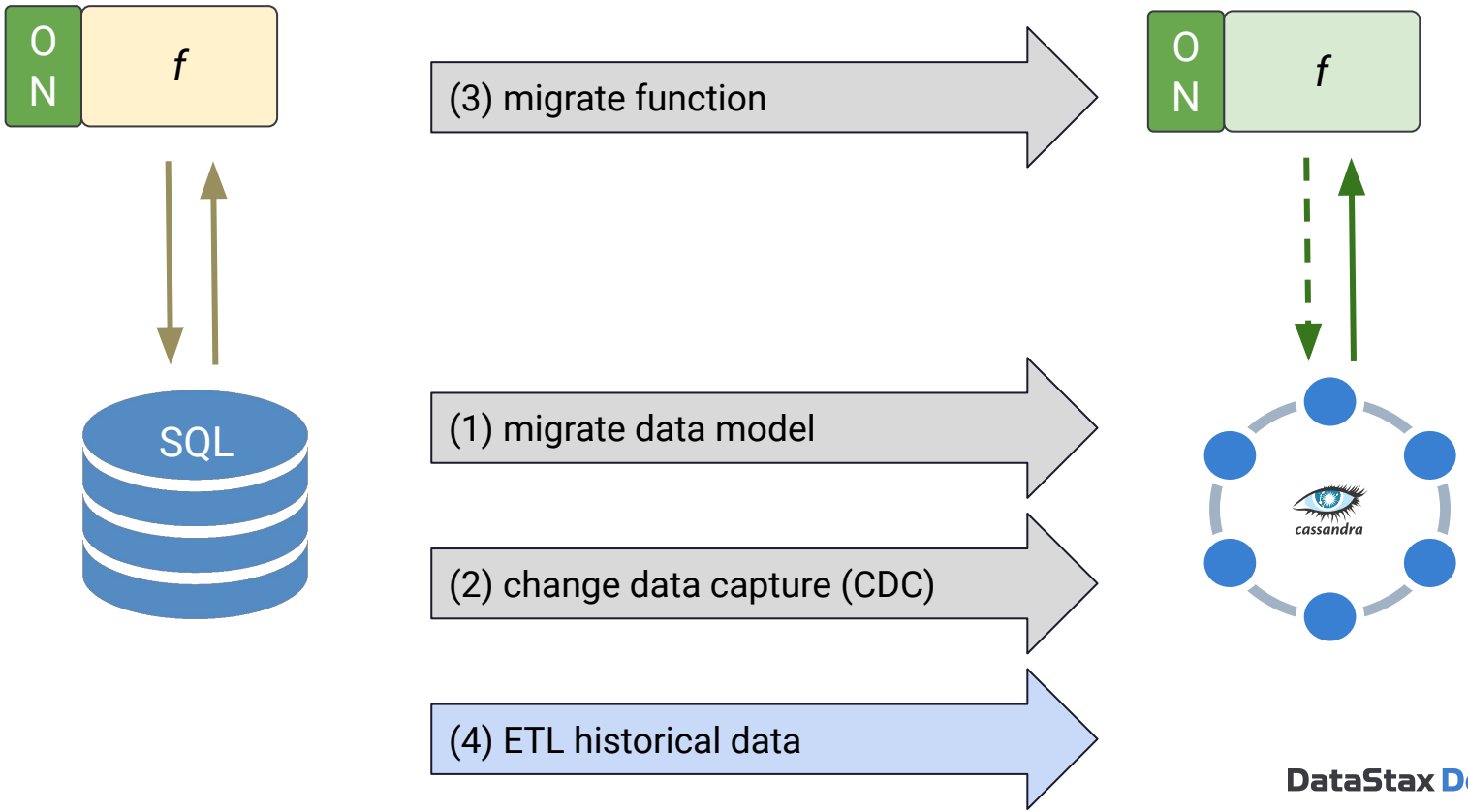
Zero-Downtime Migration with CDC

Approach 2



Zero-Downtime Migration with CDC

Approach 2



Zero-Downtime Migration with CDC

Approach 2