

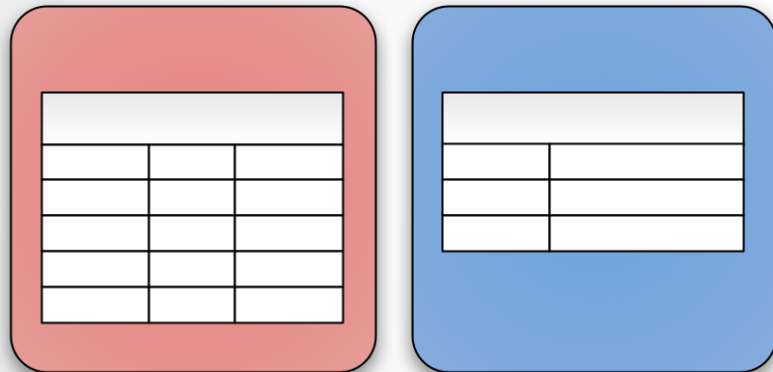


Sharding MySQL with Vitess

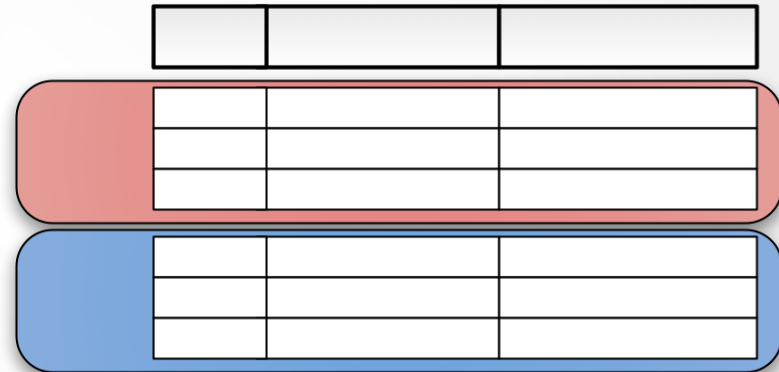
Harun Küçük

What is Sharding?

Sharding is a type of database partitioning that separates very large databases into smaller, faster and more easily managed parts called data shards.



Vertical

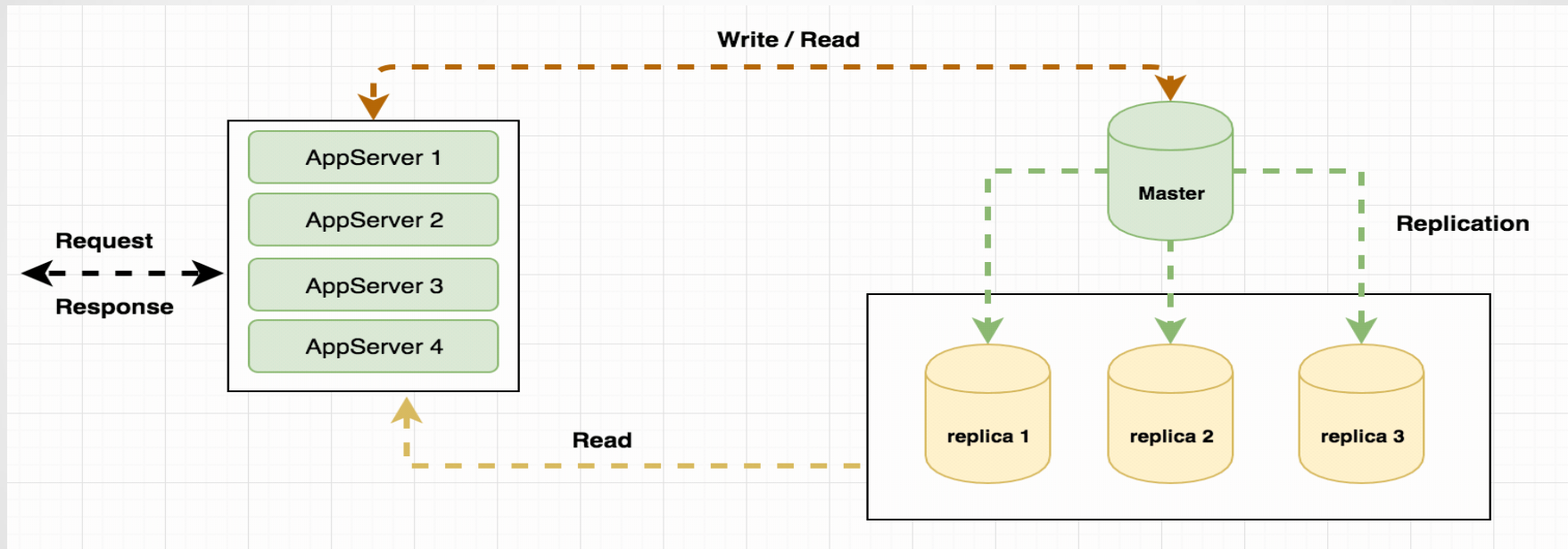


Horizontal



Why we need Sharding?

Sample Traditional MySQL Replication



- Scalable App Layer
- Scalable Replicas
- **Non-Scalable Master**



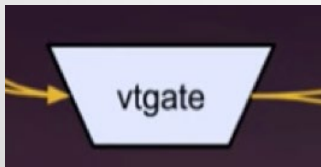
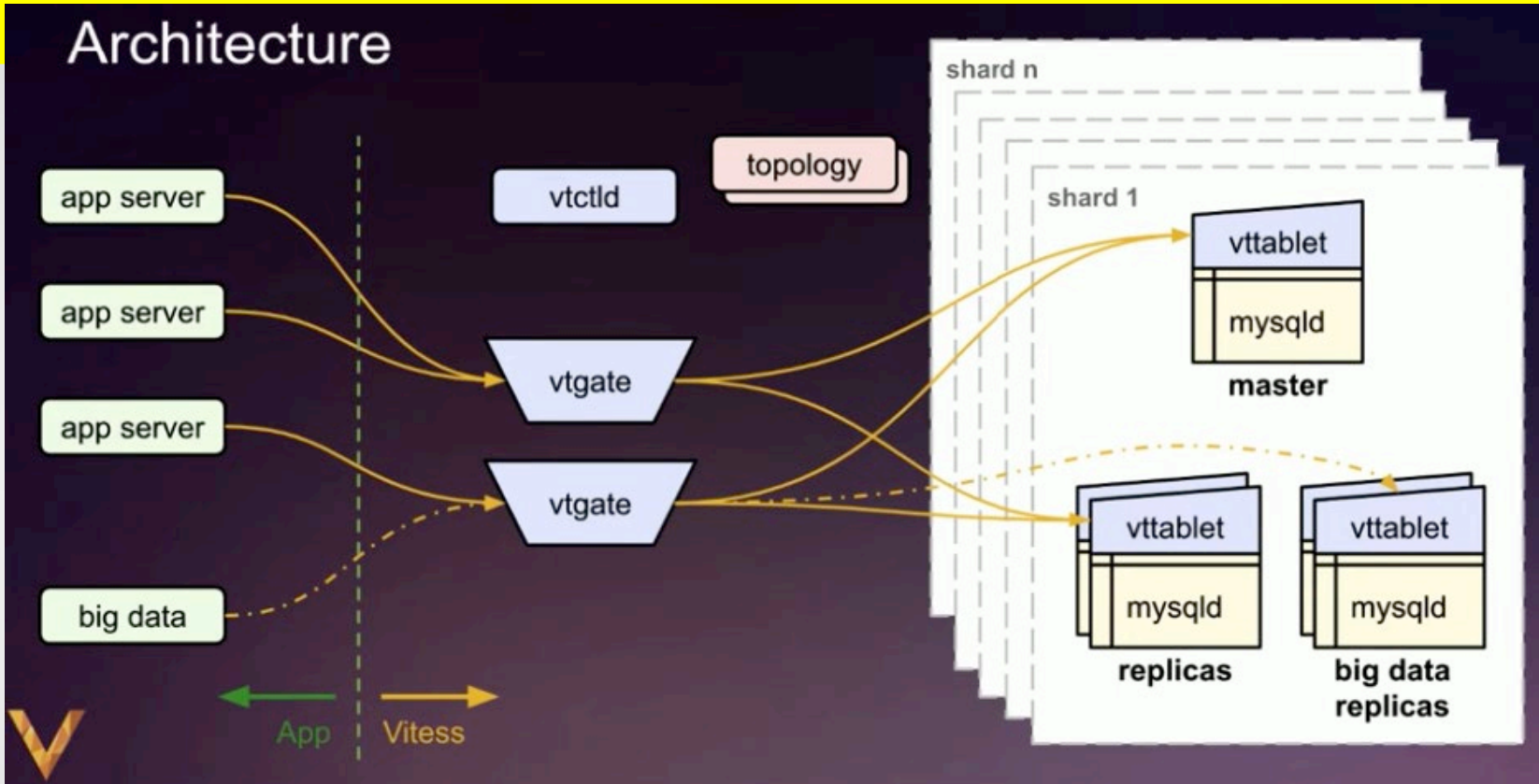
Vitess



- Started 2010 , youtube <https://vitess.io/>
- Open source since 2011 <https://github.com/vitessio/vitess>
- Incubating project in CNCF <https://www.cncf.io/projects/>



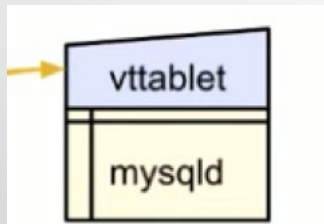
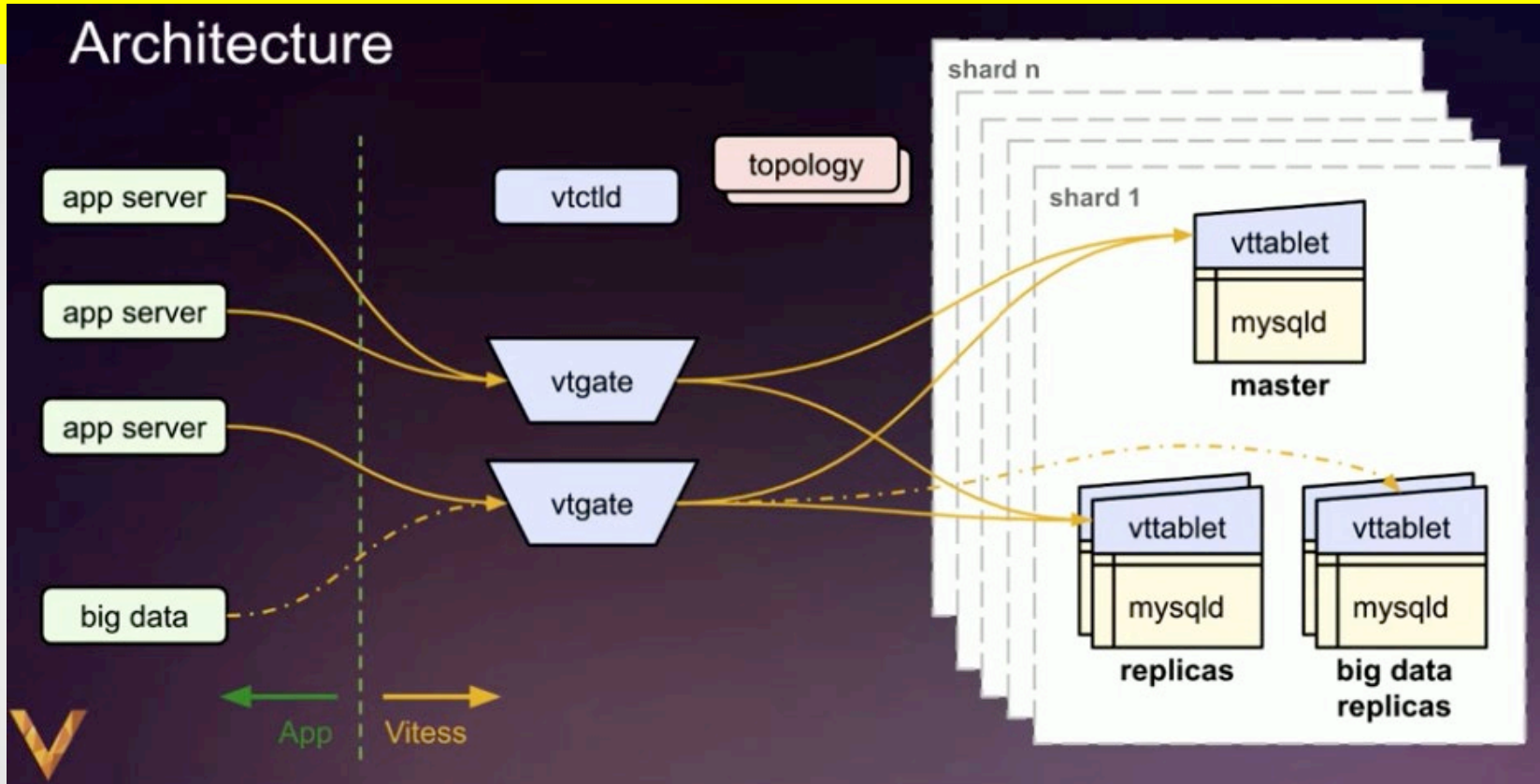
Vitess Architecture



- Lightweight proxy server
- Routes traffic to correct vtablet
- Returns consolidated results back to the clients



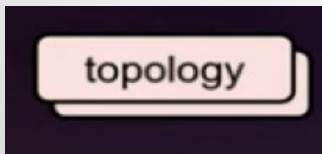
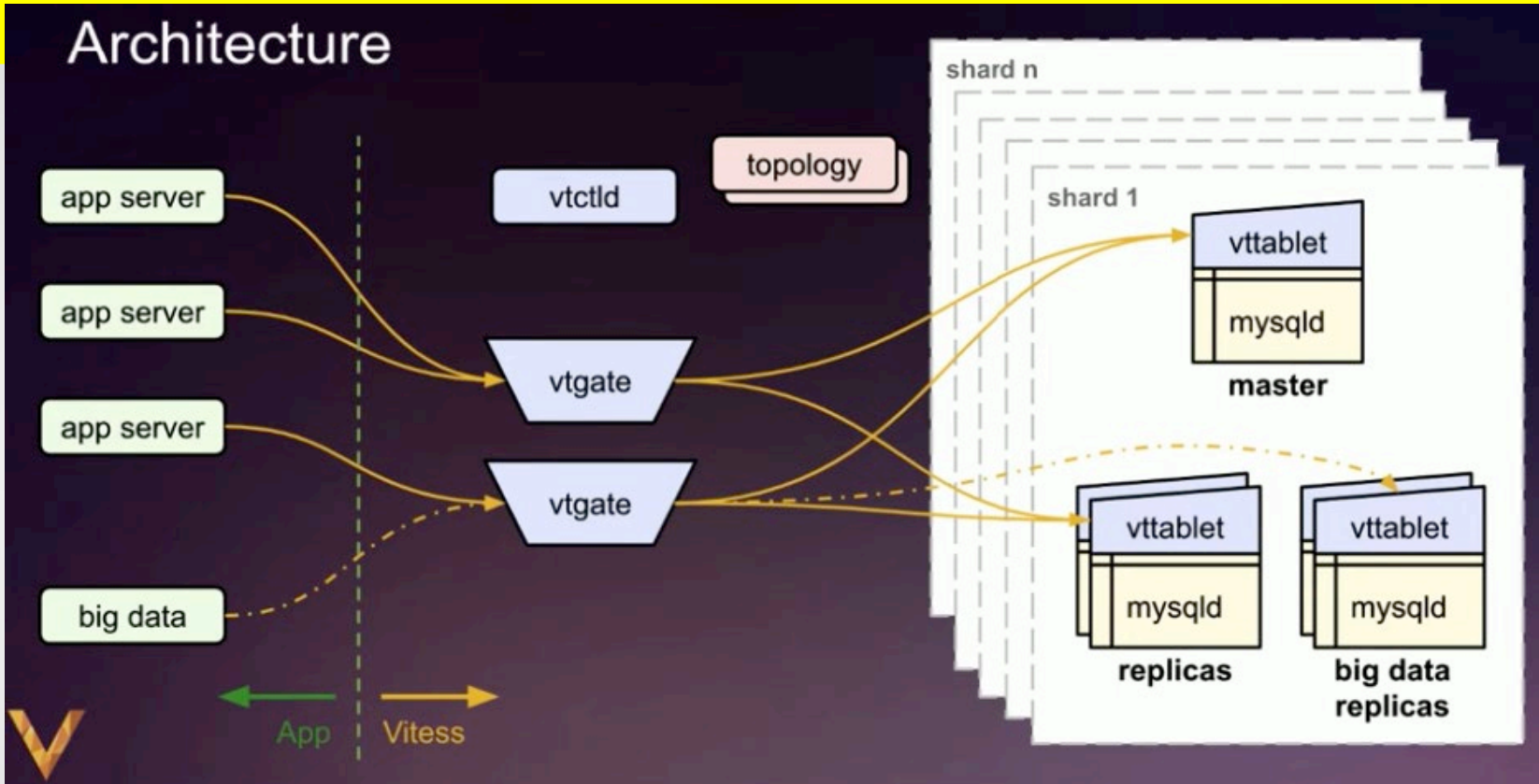
Vitess Architecture



- Proxy server that sits in front of MySQL instance
- Protect MySQL from harmful queries
- Connection Pooling
- Query rewriting
- Hot row protection



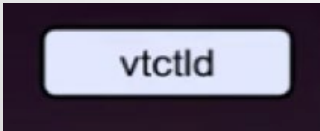
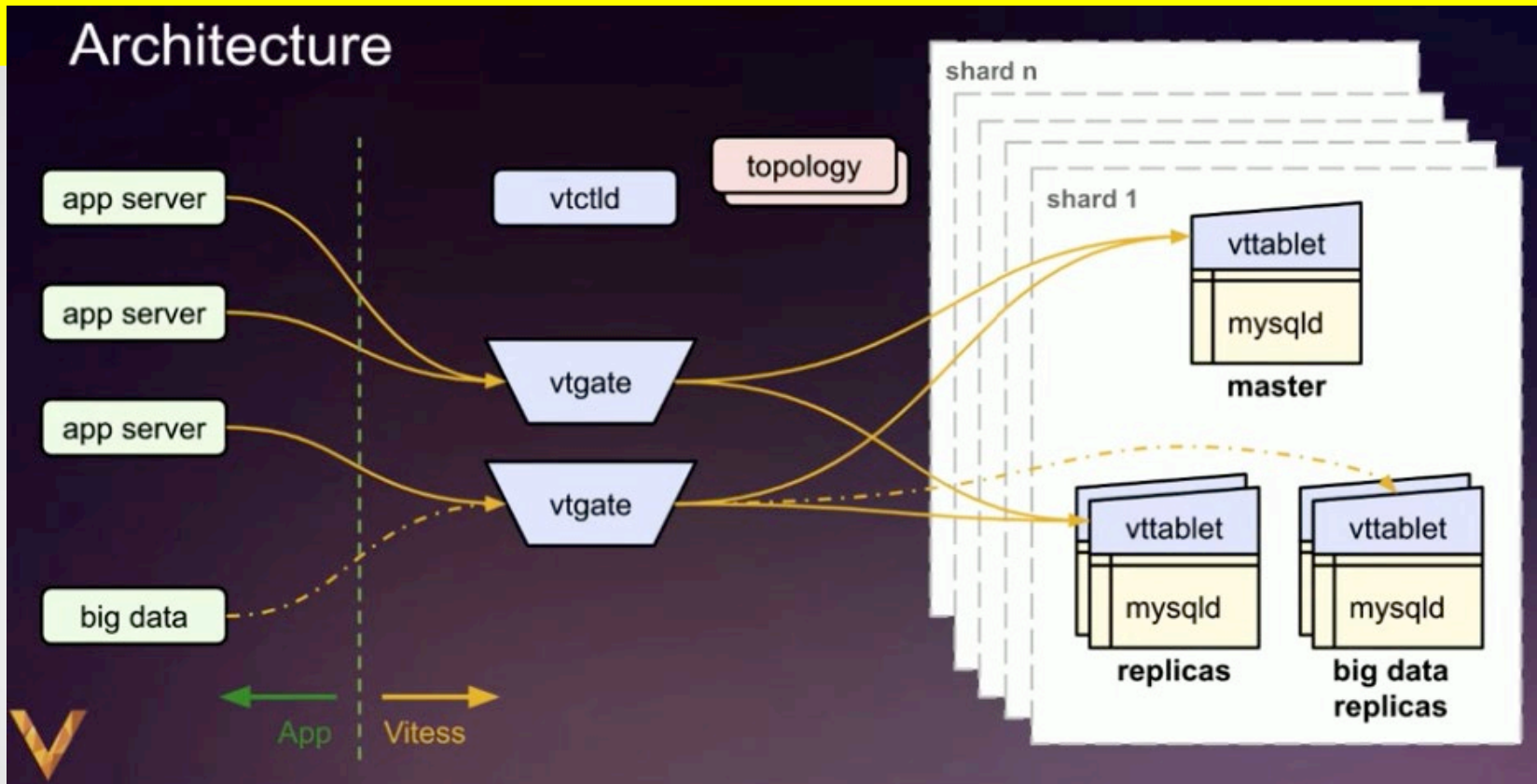
Vitess Architecture



- Stores metadata (running servers, sharding schema, Replication Graph)
- Etcd, Apache Zookeeper or consul could be used for topology



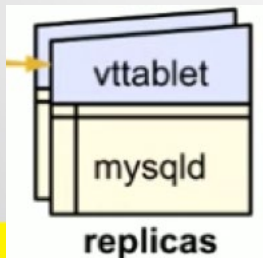
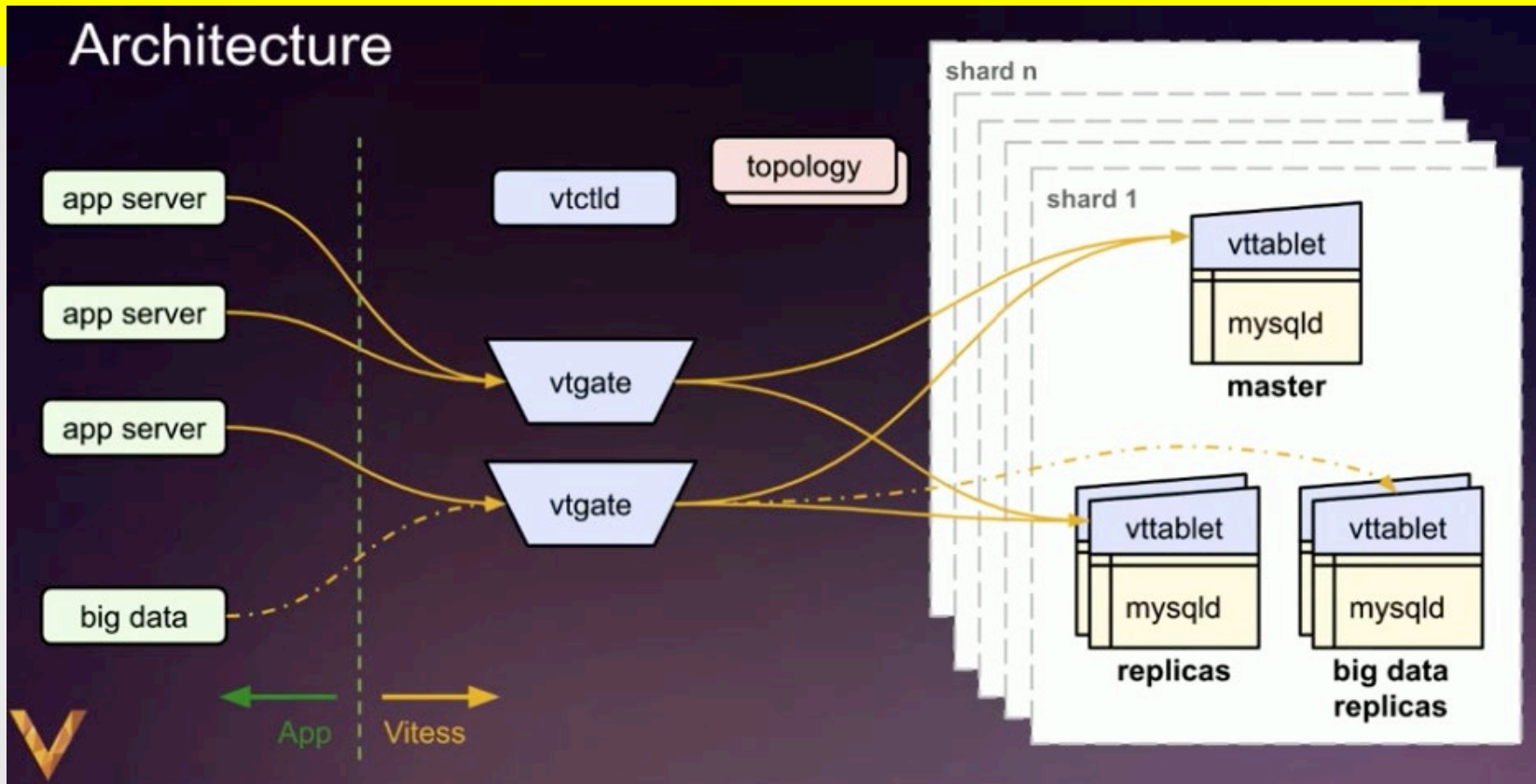
Vitess Architecture



- Vtctl is command line tool, Vtctld is an HTTP server that lets you browse the information stored in the topology.



Vitess Architecture



- **Replica tables:** candidates for master table ,
- **Readonly tables:** for batch jobs, resharding, bigdata, backups etc.



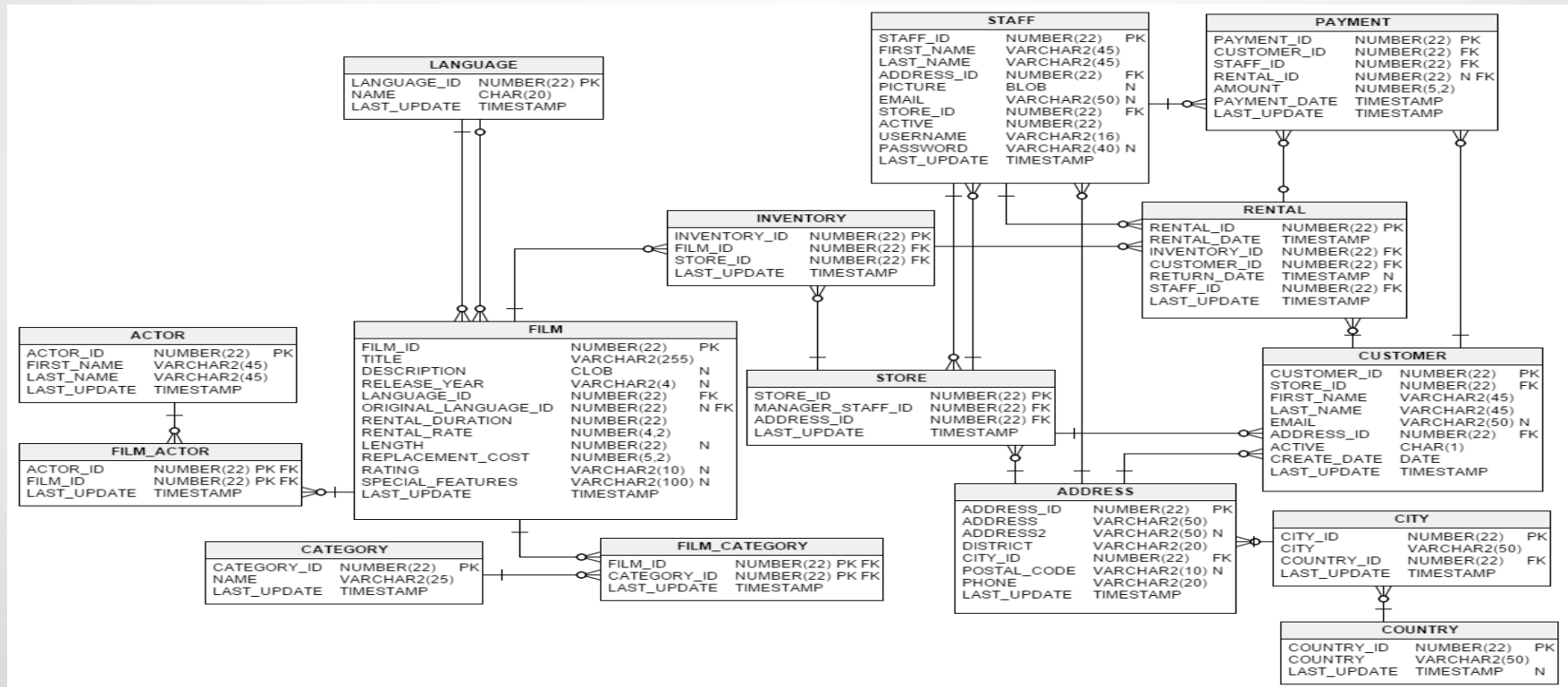
Vitess Key Adaptors

- Started 2010 at Youtube and It has been serving all Youtube database traffic since 2011. Youtube had 256 shards and each shards had between 80 and 120 replicas across 20 datacenters all around the world. (Approx. 256K instance)
- JD.com is the 2nd largest retailer company in China. JD.com has more than 10.000 instance (master,replicas) in Vitess on kubernetes cluster
- Square Cash App fully runs on Vitess. Square has more than 64 shards.
- Slack migrated 40% database traffic to Vitess and their goal is 100%
- Pinterest's all of advertising campaign management fully runs on Vitess



Example: Sakila DVD Rental Company Database

Lets suppose we have a DVD rental company and our database diagram live below



Day 1: Table Rows

Query:

```
SELECT table_name,  
       table_rows  
FROM   information_schema.tables  
WHERE  table_schema = 'sakila'  
ORDER BY table_rows DESC
```

table_name	table_rows
film_actor	5462
inventory	4581
film	1000
film_category	1000
city	600
actor	200
country	109
category	16
language	6
staff	2
store	2
payment	0
rental	0
address	0
customer	0



30 days later...

Query:

```
SELECT table_name,  
       table_rows  
FROM   information_schema.tables  
WHERE  table_schema = 'sakila'  
ORDER BY table_rows DESC
```

table_name	table_rows
payment	16081
rental	16005
film_actor	5462
inventory	4581
film	1000
film_category	1000
address	603
city	600
customer	599
actor	200
country	109
category	16
language	6
staff	2
store	2



6 months later...

Query:

```
SELECT table_name,  
       table_rows  
FROM   information_schema.tables  
WHERE  table_schema = 'sakila'  
ORDER BY table_rows DESC
```

table_name	table_rows
payment	18,042,882
rental	17,957,610
address	30,543
customer	30,543
film_actor	6,523
inventory	5,200
film	1,300
film_category	1,300
city	600
actor	300
country	109
category	16
language	6
staff	5
store	4



2 years later...

Query:

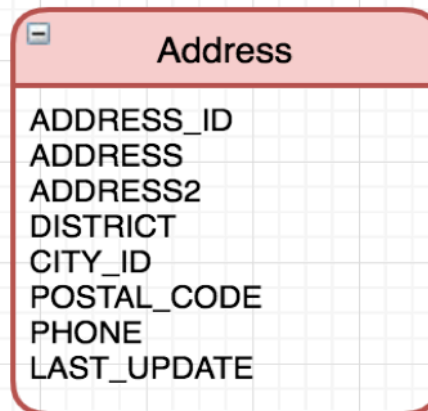
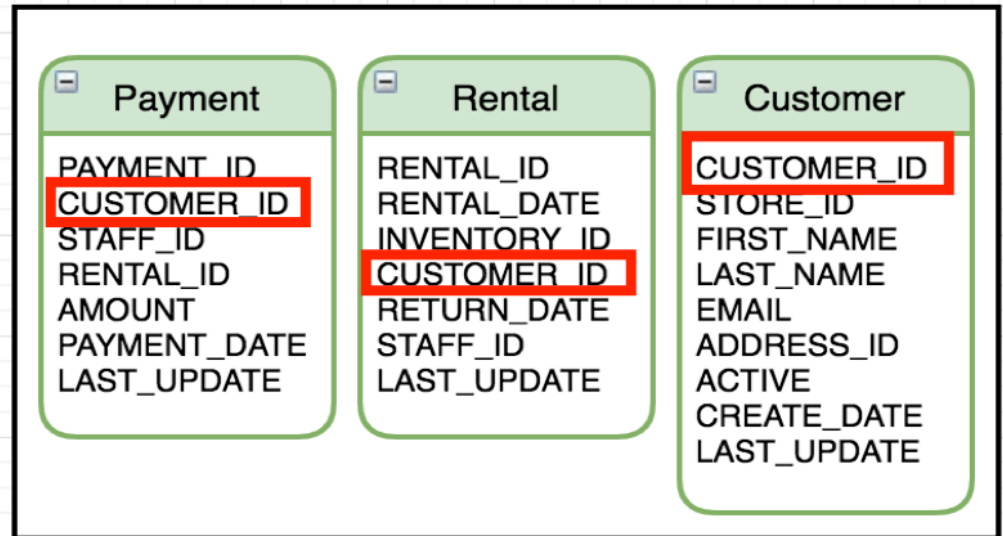
```
SELECT table_name,  
       table_rows  
FROM   information_schema.tables  
WHERE  table_schema = 'sakila'  
ORDER BY table_rows DESC
```

table_name	table_rows
payment	28,056,681,510
rental	27,924,083,550
address	31,276,032
customer	31,276,032
film_actor	9,286
inventory	8,992
film	2,543
film_category	2,543
city	600
actor	550
country	109
category	16
language	6
staff	5
store	4

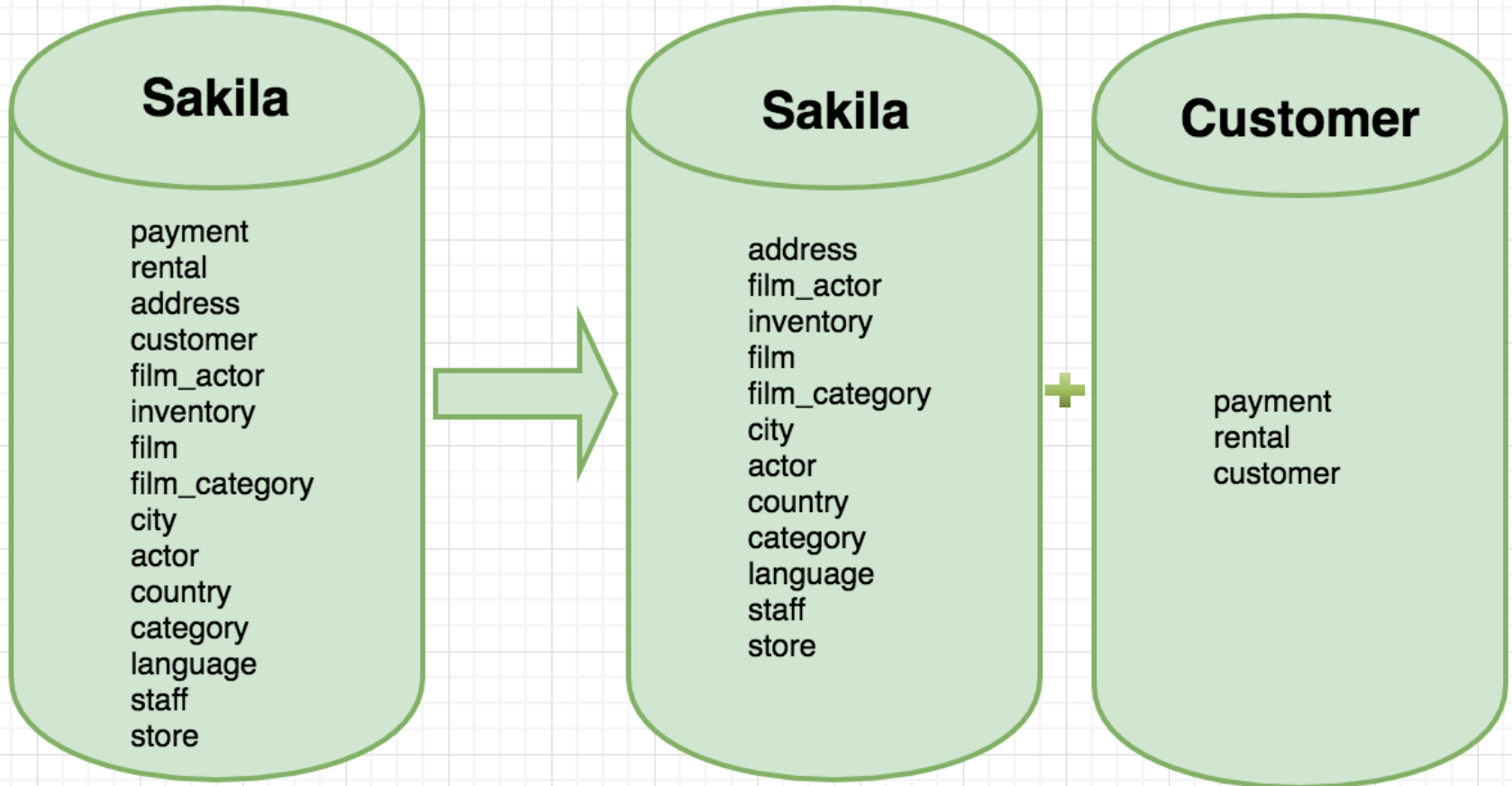


Entity Group for Sharding

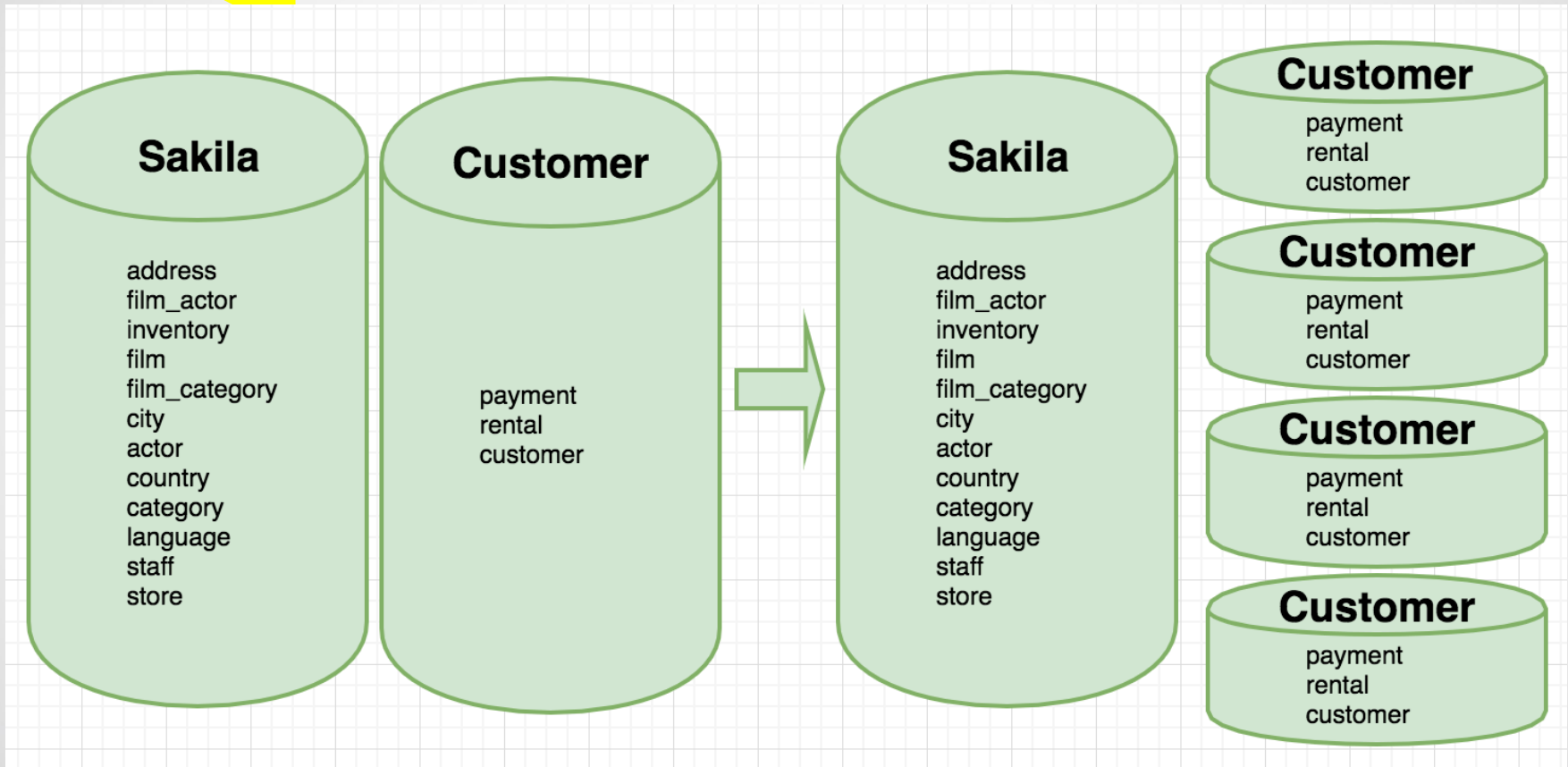
Payment, Rental and Customer tables have **customer_id** column. So these three tables should be sharded horizontally by customer_id



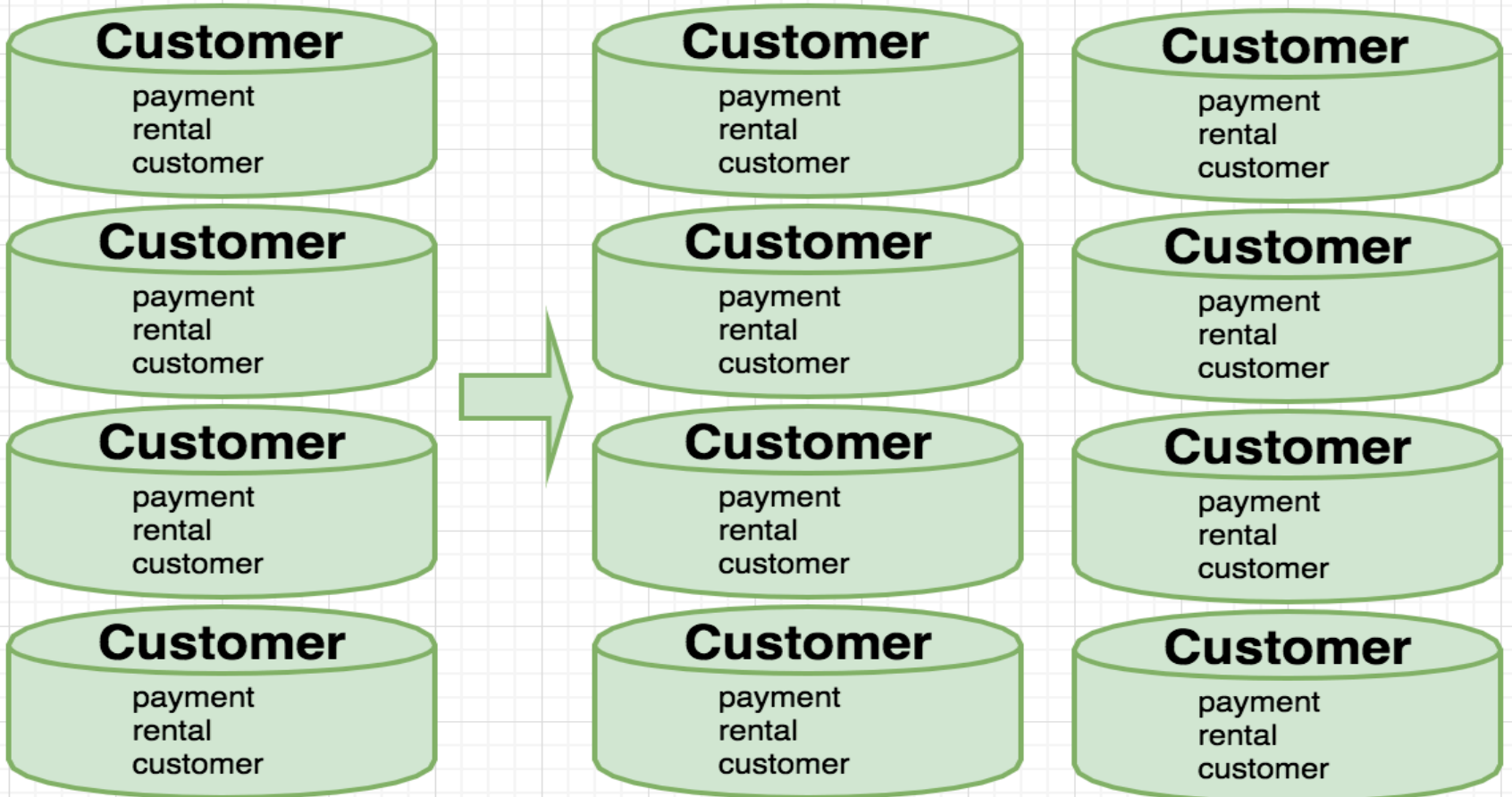
Step 1: Vertical Sharding



Step 2: Horizontal Sharding



Step 3: Resharding



Prerequisites for Demo

- Vitess (<https://github.com/vitessio/vitess>)
- Kubernetes Cluster or Minicube
- Etcd operator for topology cluster
- Helm for vitess helm charts (vitess/helm/vitess)
- NFS client provisioner for persistent NFS volumes
(<https://github.com/helm/charts/tree/master/stable/nfs-client-provisioner>)



1.) initiate new cluster, Concepts

File: initial_cluster.yaml

- **Cell** : Zone, availability zone, datacenter
- **KeySpace** : Logical Database
- **Vschema** : Vitess Schema, contains metadata about how tables are organized across keyspaces and shards
- **Vindex** : index to find shards

```
cells:  
  - name: "sakiladb"  
    etcd:  
      replicas: 3  
    vtctld:  
      replicas: 1  
    vtgate:  
      replicas: 3
```

```
keyspaces:  
  - name: "sakila"  
    shards:  
      - name: "0"  
        tablets:  
          - type: "replica"  
            vtablet:  
              replicas: 2  
          - type: "rdonly"  
            vtablet:  
              replicas: 1
```

```
vschema:  
  initial: |-  
    {  
      "tables": {  
        "actor": {},  
        "address": {},  
        "category": {},  
        "city": {},
```



1.) initiate new cluster

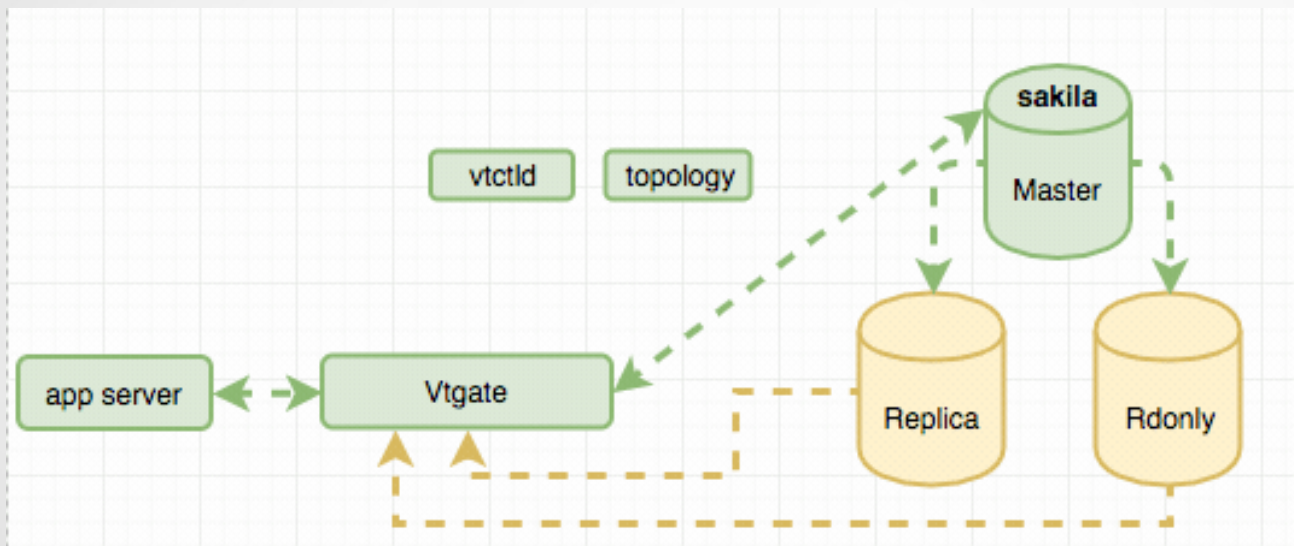
File: initial_cluster.yaml

```
root@node1:~/sakiladb#
```

```
|
```



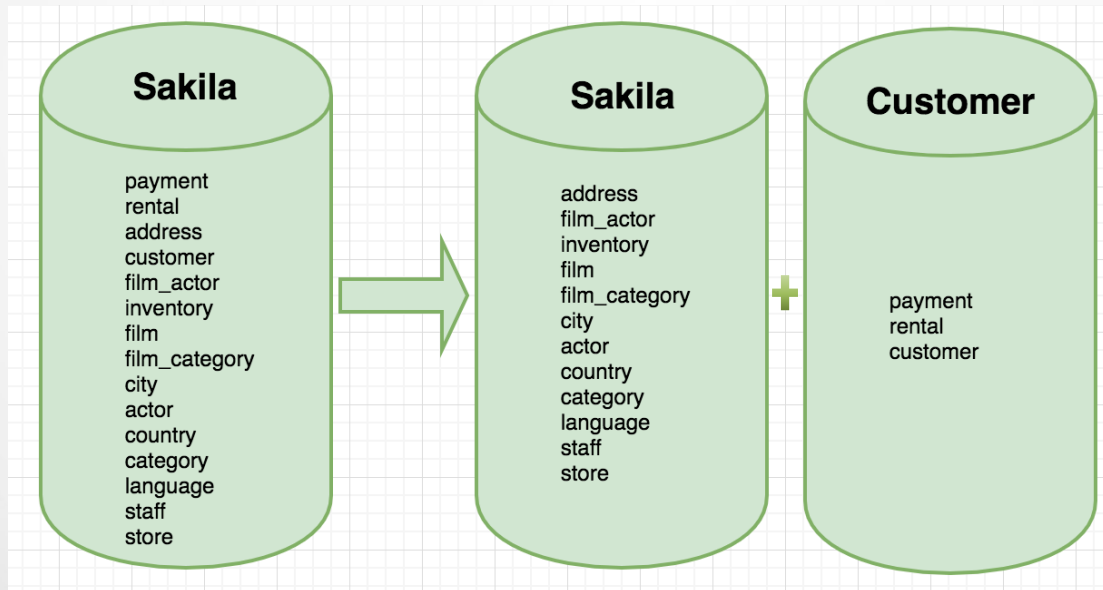
1.) initiate new cluster



2.) Create New Keyspace

File:create_keyspace.yaml

```
jobs:  
- name: "create-customer-keyspace"  
  kind: "vtctlclient"  
  command: "CreateKeyspace -served_from='master:sakila,replica:sakila,rndonly:sakila' customer"
```



2.) Create New Keyspace

File:create_keyspace.yaml

```
root@node1:~/sakiladb#
```

```
|
```



3.) Split Keyspace Schema

File:split_keyspace_schema.yaml

```
- name: "customer"
  shards:
    - name: "0"
      tablets:
        - type: "replica"
          vtablet:
            replicas: 2
        - type: "ronly"
          vtablet:
            replicas: 1
      copySchema:
        source: "sakila/0"
        tables:
          - "customer"
          - "payment"
          - "rental"
      vschema:
        vsplit: |-
          {
            "tables": {
              "customer": {},
              "payment": {},
              "rental": {}
            }
          }
```



3.) Split Keyspace Schema

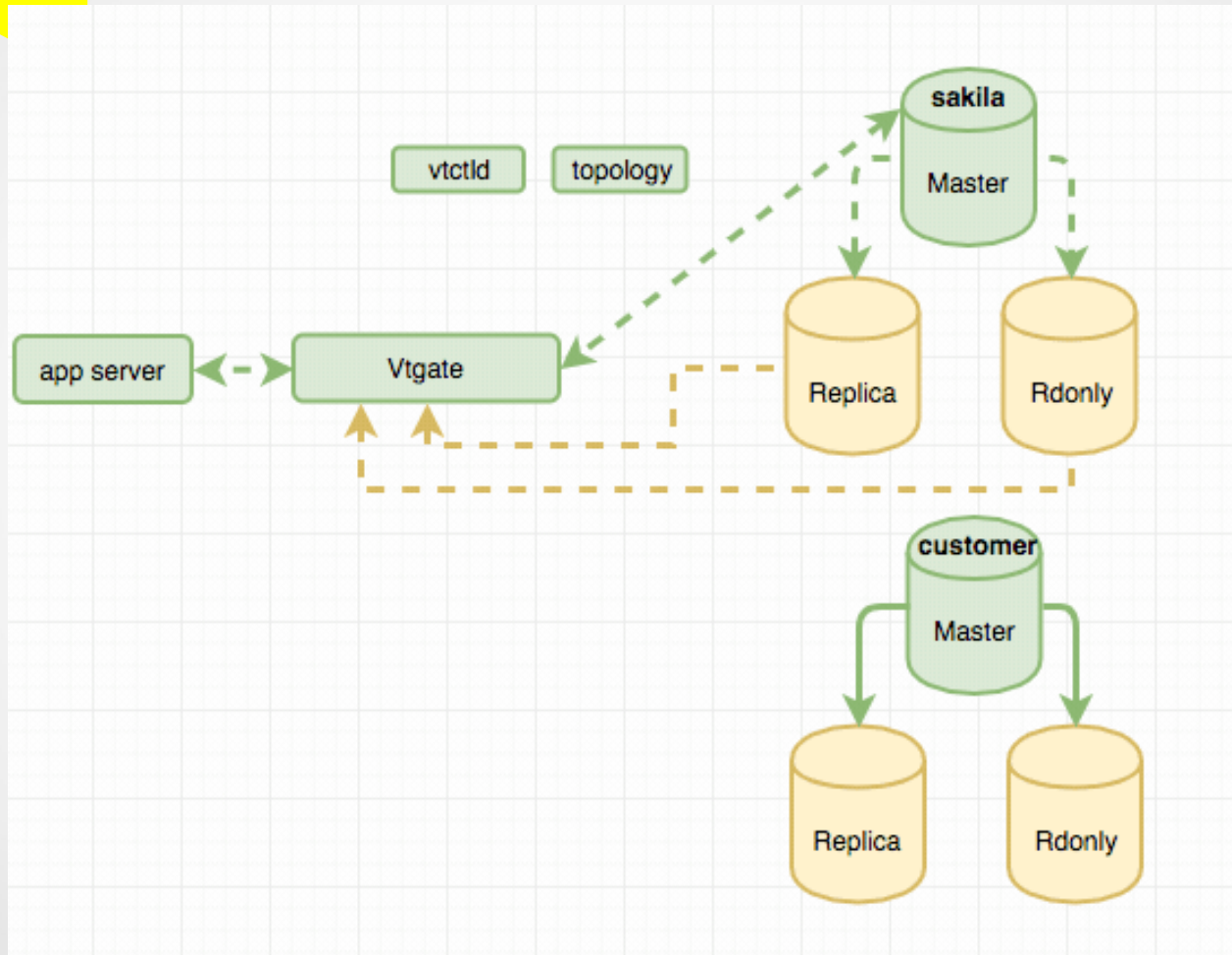
File:split_keyspace_schema.yaml

```
root@node1:~/sakiladb#
```

```
|
```



3.) Split KeySpace Schema



4.) Vertical Split Clone

File:vertical_split_clone.yaml

```
jobs:  
  - name: "verticalsplitclone"  
    kind: "vtworker"  
    cell: "sakiladb"  
    command: "VerticalSplitClone -min_healthy_ronly_tablets=1 -tables=customer,payment,rental customer/0"
```



4.) Vertical Split Clone

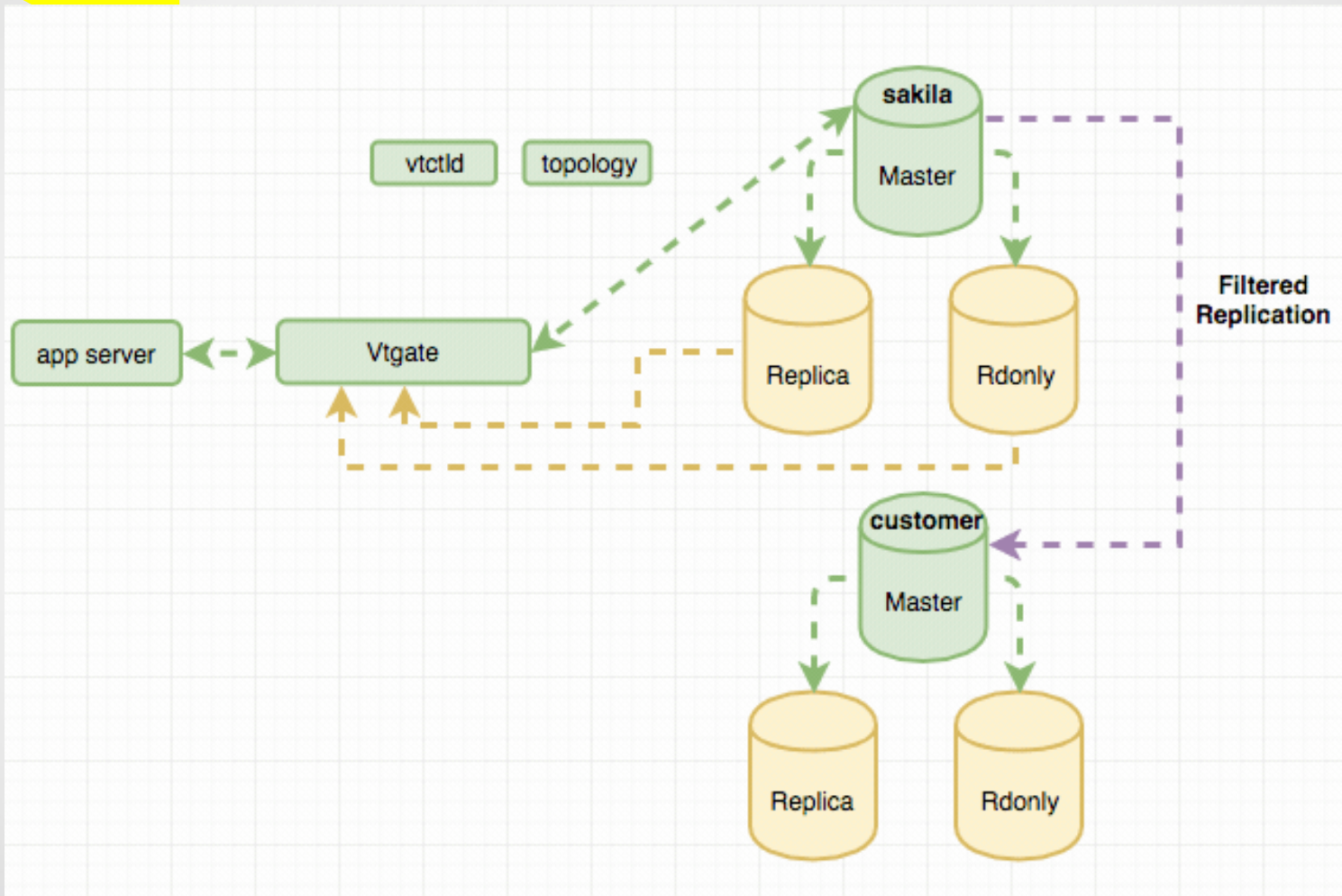
File:vertical_split_clone.yaml

```
root@node1:~/sakiladb#
```

```
|
```



4.) Vertical Split Clone



4.) Migrate

File: migrate_readonly_replica.yaml
migrate_master.yaml

```
jobs:  
- name: "migratereadonly"  
  kind: "vtctlclient"  
  command: "MigrateServedFrom customer/0 ronly"  
- name: "migratereplica"  
  kind: "vtctlclient"  
  command: "MigrateServedFrom customer/0 replica"
```

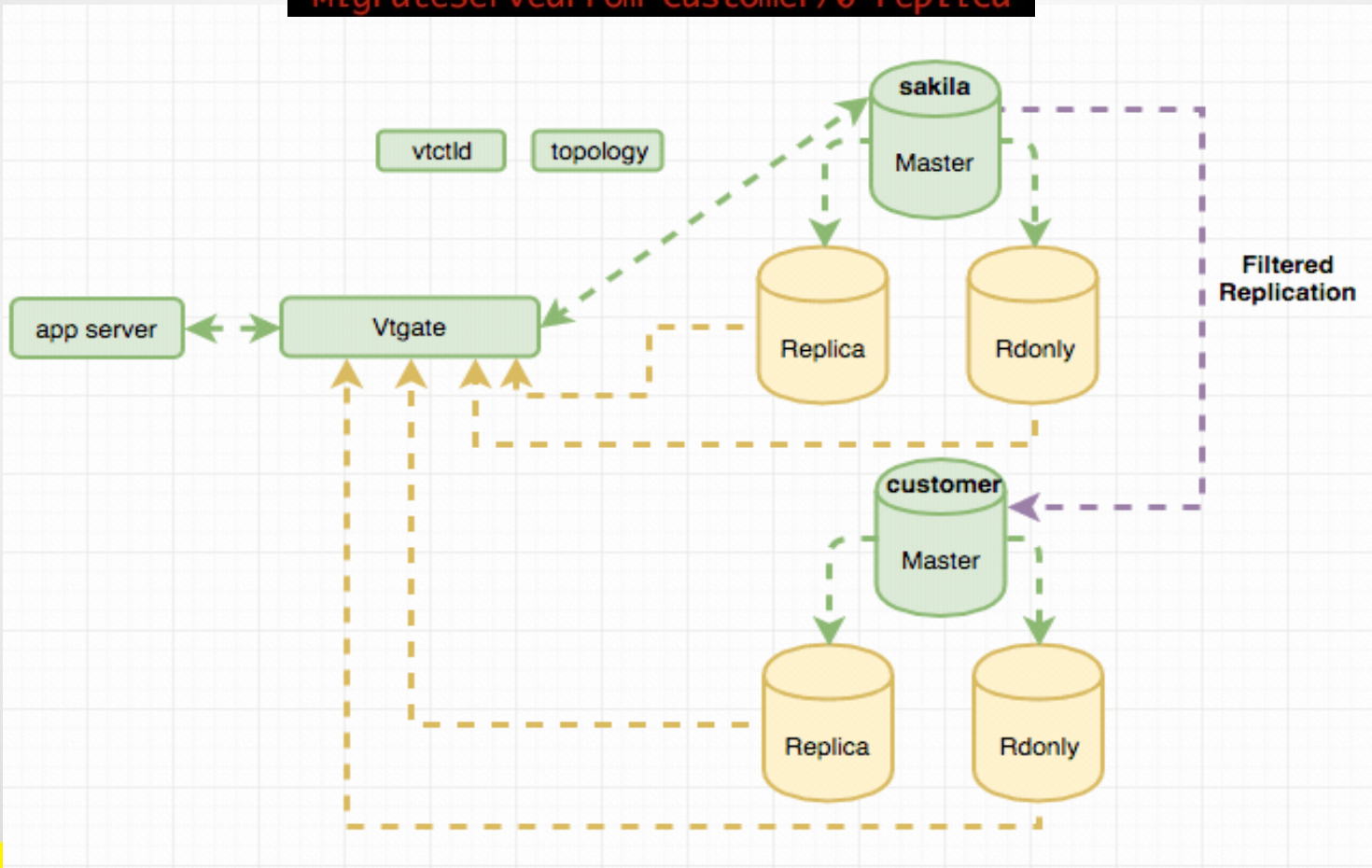
```
jobs:  
- name: "migratemaster"  
  kind: "vtctlclient"  
  command: "MigrateServedFrom customer/0 master"
```



4.) Migrate ReadOnly and Replica

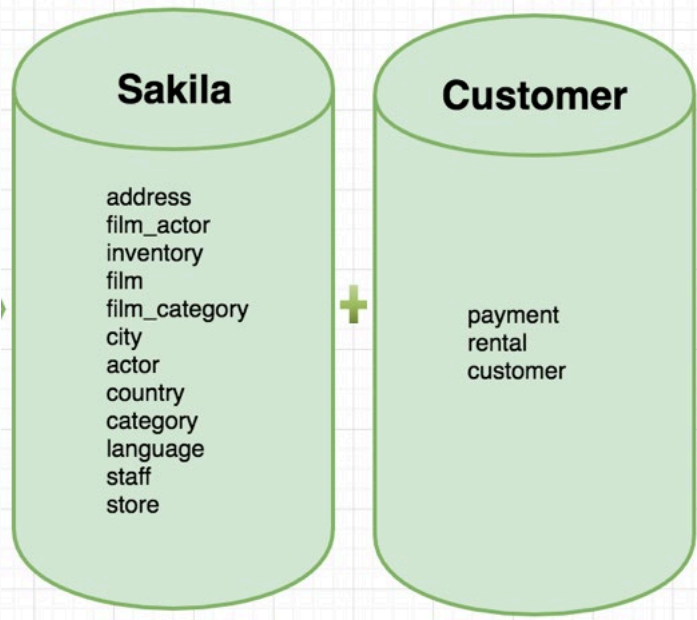
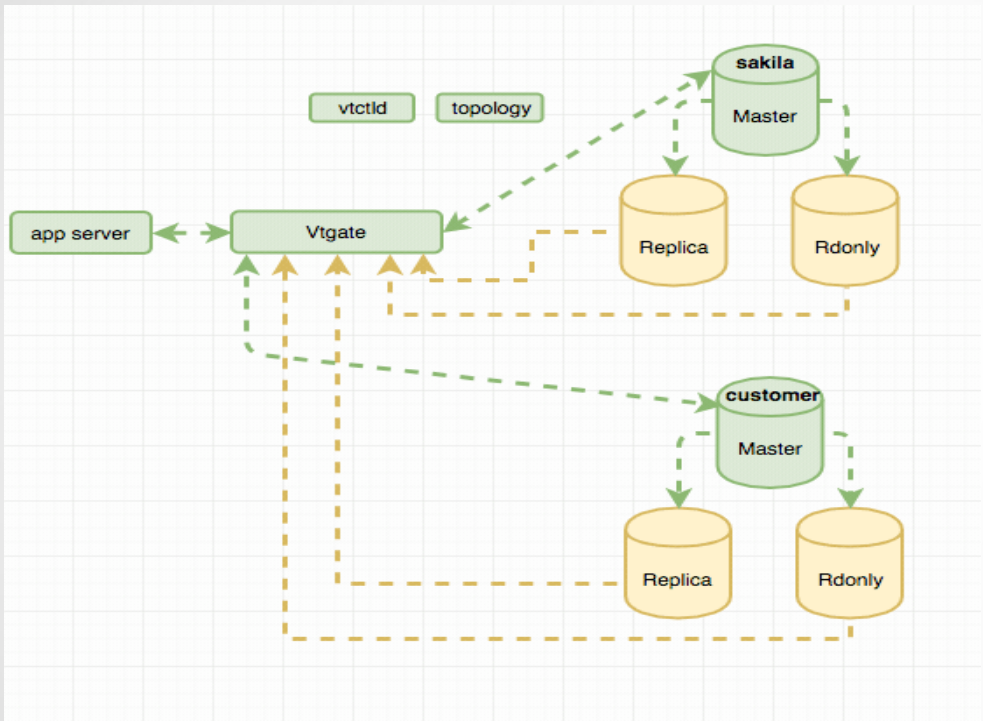
"MigrateServedFrom customer/0 ronly"

"MigrateServedFrom customer/0 replica"



4.) Migrate Master

```
"MigrateServedFrom customer/0 master"
```



5.) Drop Blacklisted Tables

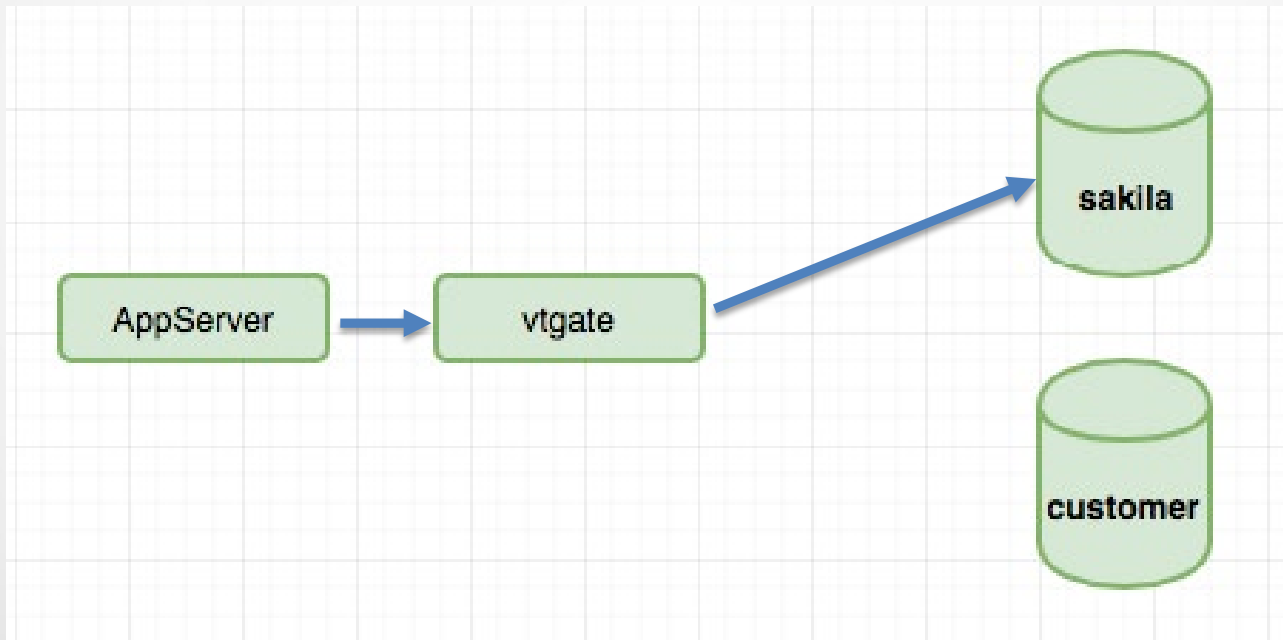
```
schema:  
  postsplit: |-  
    drop table customer;  
    drop table payment;  
    drop table rental;
```

```
jobs:  
- name: "cleanreadonly"  
  kind: "vtctlclient"  
  command: "SetShardTabletControl -blacklisted_tables=customer,payment,rental -remove sakila/0 ronly"  
- name: "cleanreplica"  
  kind: "vtctlclient"  
  command: "SetShardTabletControl -blacklisted_tables=customer,payment,rental -remove sakila/0 replica"  
- name: "cleanmaster"  
  kind: "vtctlclient"  
  command: "SetShardTabletControl -blacklisted_tables=customer,payment,rental -remove sakila/0 master"
```



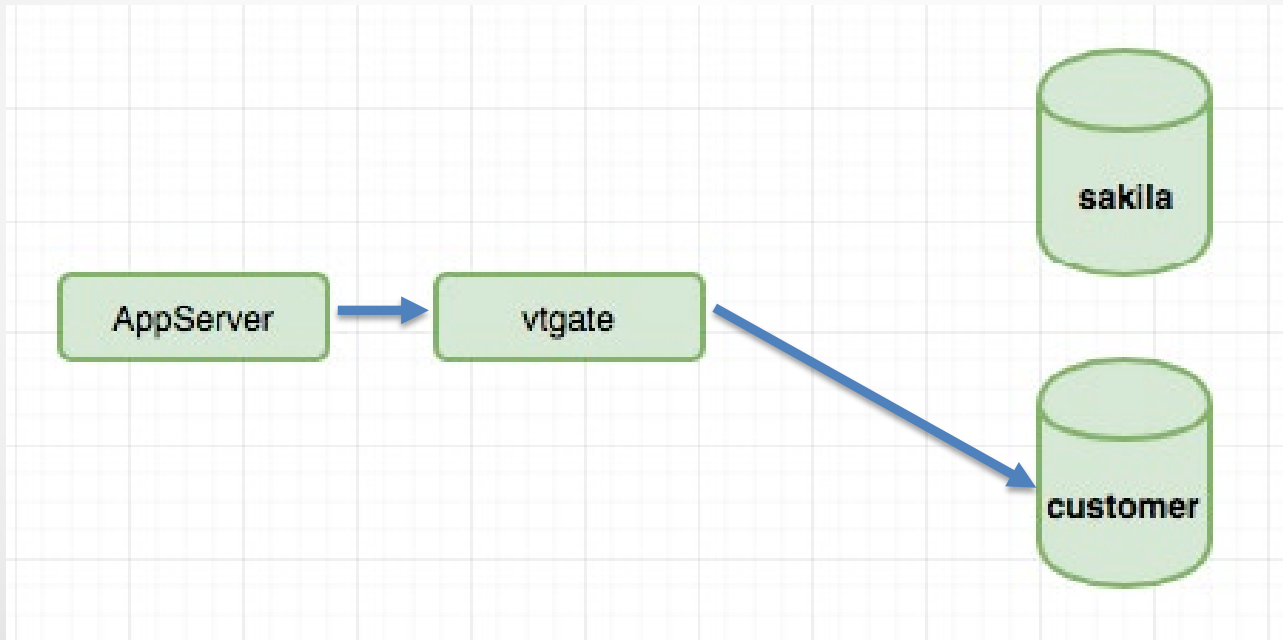
VtGate Query Routing

```
#Query 1  
SELECT *  
FROM   staff  
WHERE  staff_id=2;
```

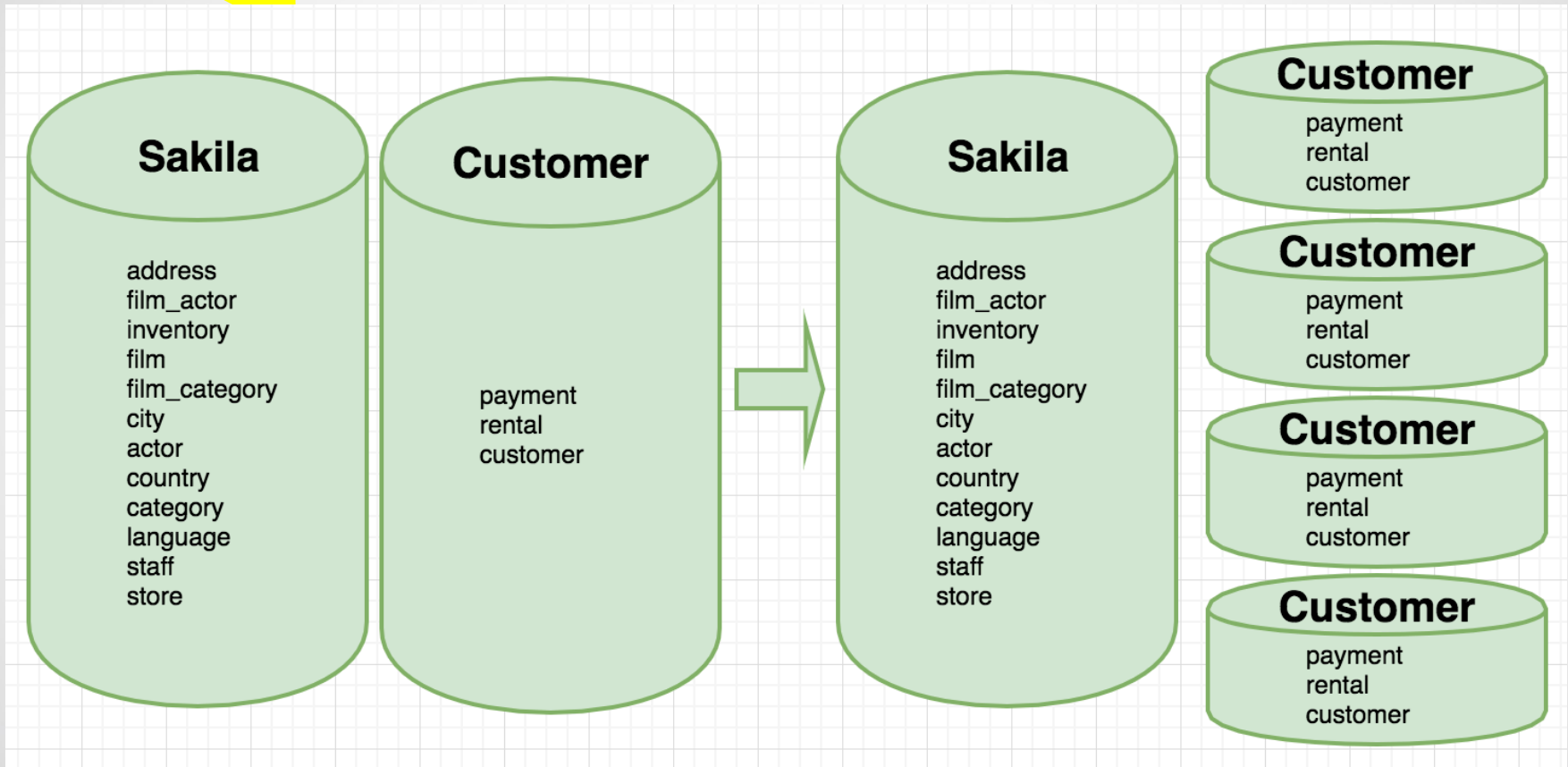


VtGate Query Routing

```
#Query 2  
SELECT *  
FROM payment  
WHERE payment_id=5;
```



Horizontal Sharding



6.) Primary Vindex for Horizontal Sharding

File: create_vindex.yaml

Payment	Rental	Customer
PAYMENT_ID	RENTAL_ID	CUSTOMER_ID
CUSTOMER_ID	RENTAL_DATE	STORE_ID
STAFF_ID	INVENTORY_ID	FIRST_NAME
RENTAL_ID	CUSTOMER_ID	LAST_NAME
AMOUNT	RETURN_DATE	EMAIL
PAYMENT_DATE	STAFF_ID	ADDRESS_ID
LAST_UPDATE	LAST_UPDATE	ACTIVE
		CREATE_DATE
		LAST_UPDATE

```
vschema:  
  sharded: |-  
    {  
      "sharded": true,  
      "vindexes": {  
        "hash": {  
          "type": "hash"  
        }  
      }  
    },
```

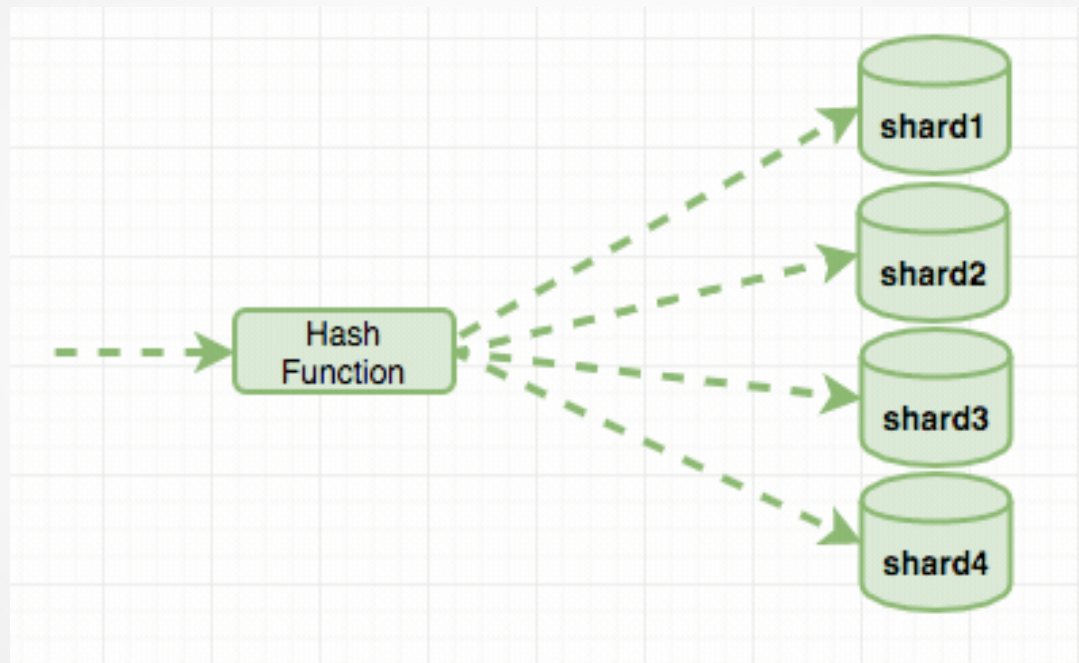
```
"payment": {  
  "column_vindexes": [  
    {  
      "column": "customer_id",  
      "name": "hash"  
    }  
  ],
```

```
"rental": {  
  "column_vindexes": [  
    {  
      "column": "customer_id",  
      "name": "hash"  
    }  
  ],
```

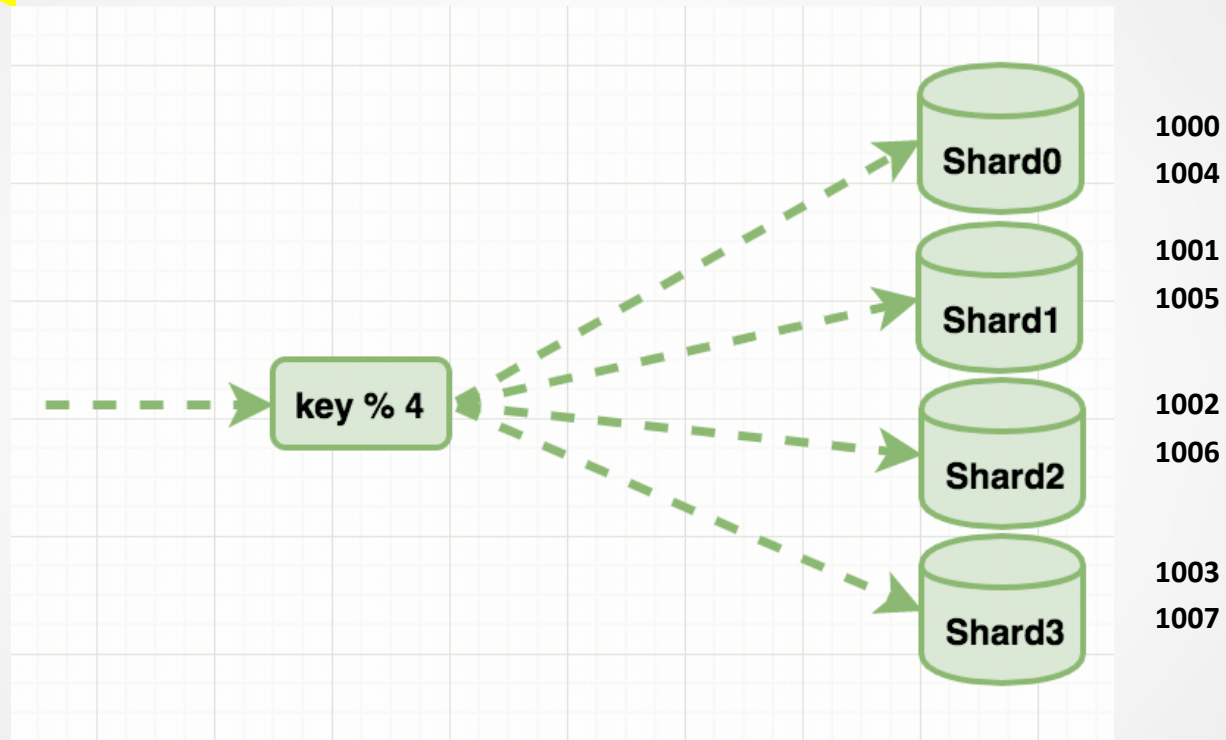
```
"customer": {  
  "column_vindexes": [  
    {  
      "column": "customer_id",  
      "name": "hash"  
    }  
  ],
```



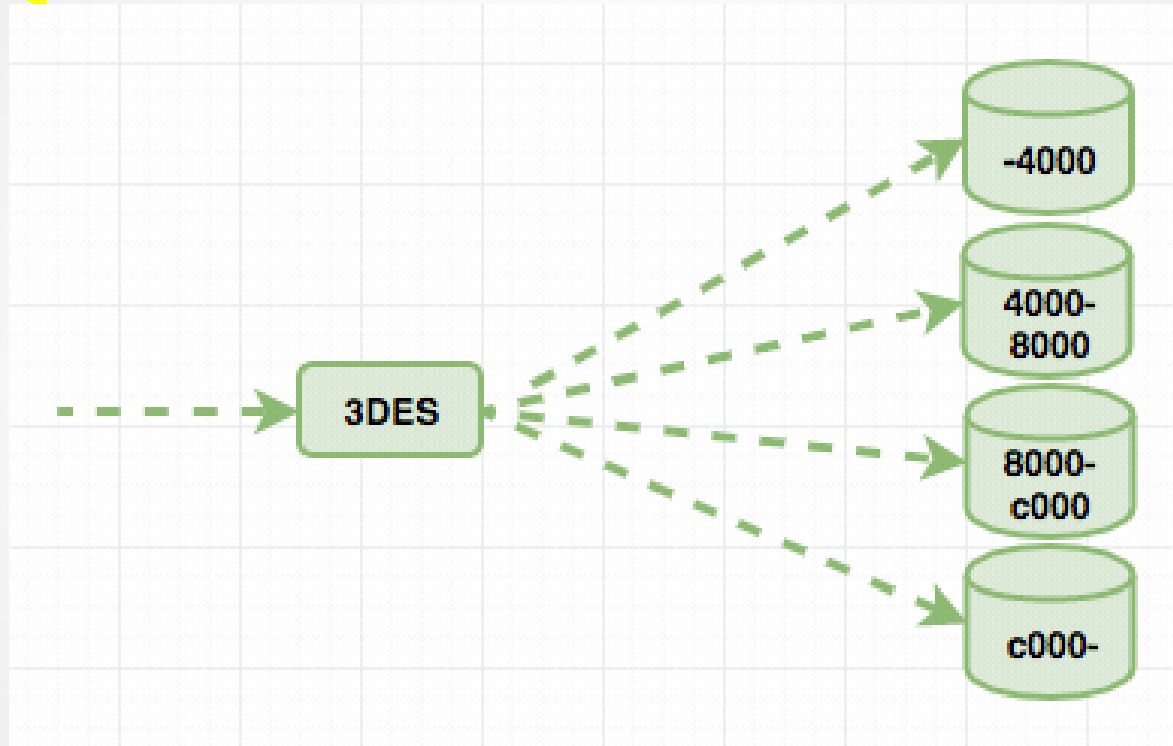
Sharding by Hash Function



Sharding by Hash Function



Vitess Hash Algorithm (3des hash)



4000 = 400000000000000000

8000 = 800000000000000000



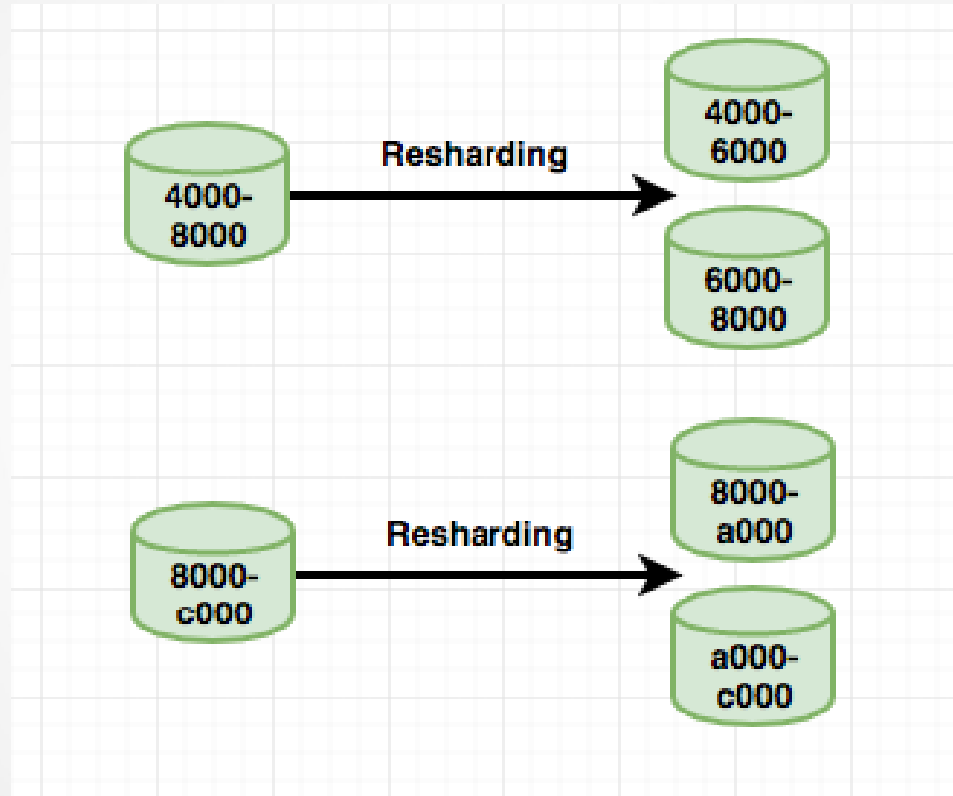
Vitess Hash Based Sharding (3des hash)

-4000	{	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4000-8000	{	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
		5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
		6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
		7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8000-C000	{	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
		9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
		A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
		B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
C000-	{	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
		D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
		E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
		F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

2^{64} combination. theoretically allows us to have an infinite number of shards



Vitess Hash Based Sharding (3des hash)



7.) Create Horizontal Shards

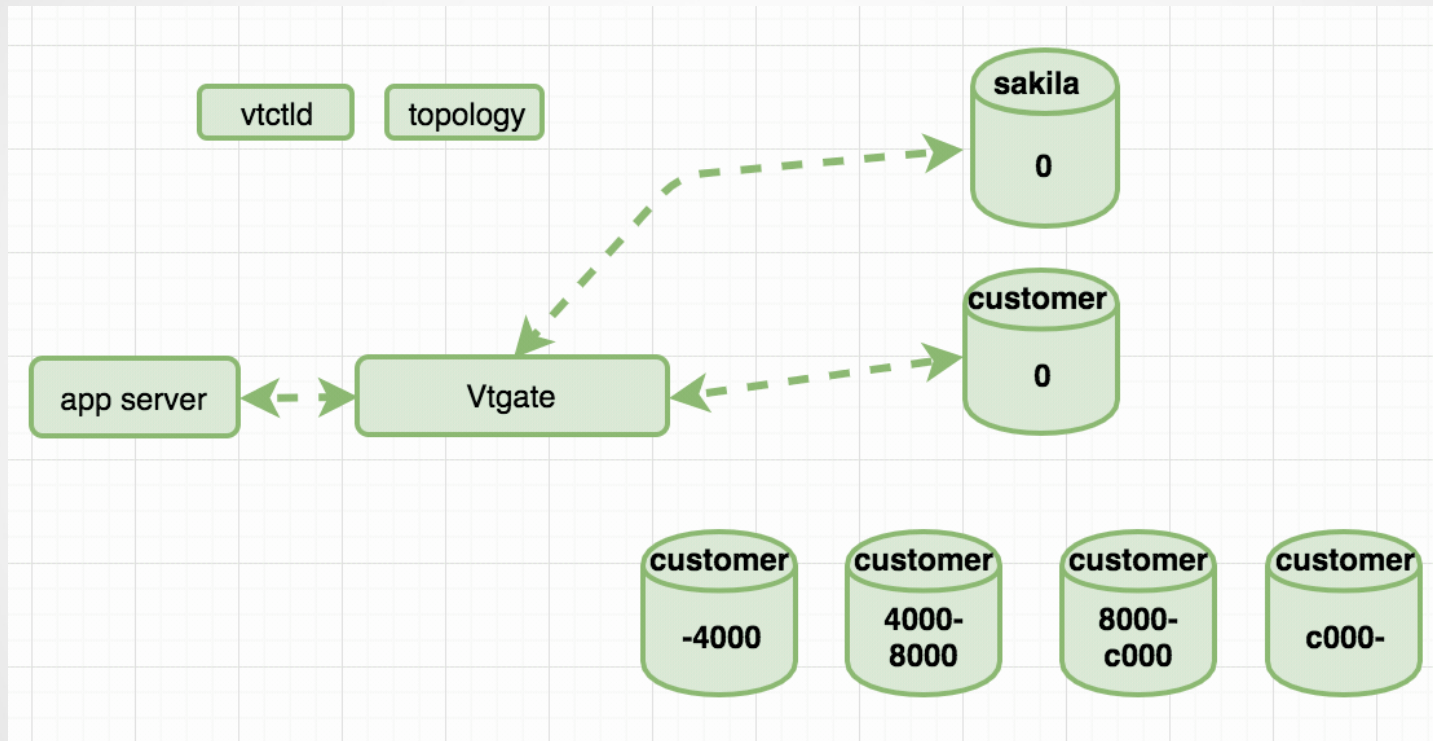
File: create_horizontal_shards.yaml

```
root@node1:~/sakiladb#
```

```
|
```



7.) Create Horizontal Shards



7.) Horizontal SplitClone

File: horizontal_split_clone.yaml

```
jobs:  
  - name: "horizontal-split"  
    kind: "vtworker"  
    cell: "sakiladb"  
    command: "SplitClone -min_healthy_ronly_tablets=1 customer/0"
```



7.) Horizontal SplitClone

File: horizontal_split_clone.yaml

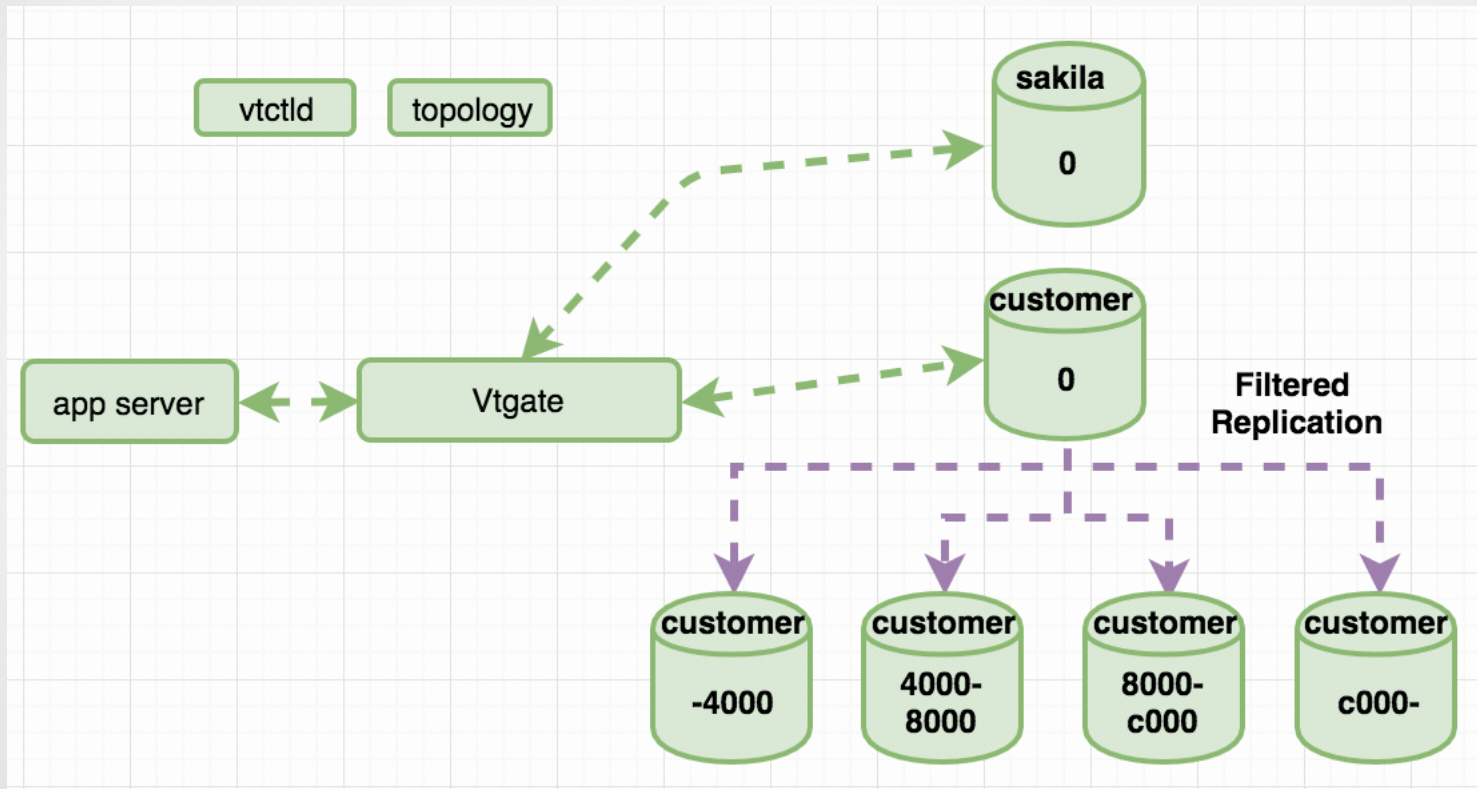
```
root@node1:~/sakiladb#
```

```
|
```



7.) Horizontal SplitClone

File: horizontal_split_clone.yaml



8.) Migrate RdOnly, Replica and Master

File: 13_migrate_readonly_replica.yaml
14_migrate_master.yaml

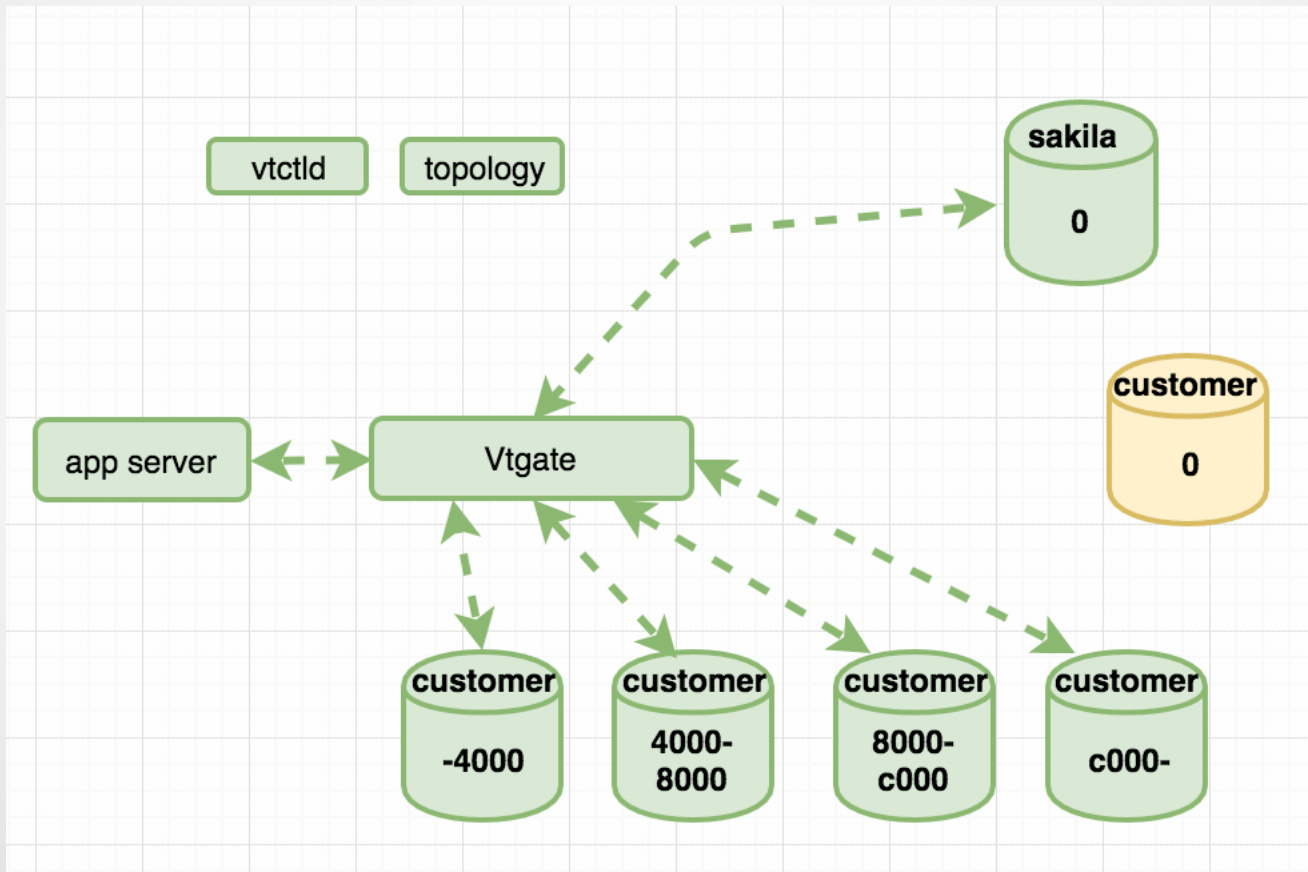
```
jobs:  
- name: "migraterdonly"  
  kind: "vtctlclient"  
  command: "MigrateServedTypes customer/0 rdonly"  
- name: "migratereplica"  
  kind: "vtctlclient"  
  command: "MigrateServedTypes customer/0 replica"
```

```
jobs:  
- name: "migratemaster"  
  kind: "vtctlclient"  
  command: "MigrateServedTypes customer/0 master"
```



8.) Migrate RdOnly, Replica and Master

File: 13_migrate_readonly_replica.yaml
14_migrate_master.yaml



Querying Vitess Cluster - Counts

```
root@node1:~/sakiladb#
```

```
|
```



Querying Vitess Cluster - Routing

```
SELECT * FROM payment WHERE customer_id = 3;
```

```
SELECT * FROM payment WHERE customer_id = 11;
```

```
SELECT * FROM payment WHERE customer_id IN ( 11, 3 );
```

```
SELECT * FROM payment WHERE customer_id = 3 AND payment_id = 10;
```

```
SELECT * FROM payment WHERE customer_id = 3 OR payment_id = 10;
```

```
SELECT p.payment_id,  
       r.rental_id,  
       c.customer_id  
FROM   customer c  
       JOIN payment p  
         ON c.customer_id = p.customer_id  
       JOIN rental r  
         ON c.customer_id = r.customer_id  
WHERE  c.customer_id IN ( 3 );
```

```
SELECT p.payment_id,  
       r.rental_id,  
       c.customer_id  
FROM   customer c  
       JOIN payment p  
         ON c.customer_id = p.customer_id  
       JOIN rental r  
         ON c.customer_id = r.customer_id  
WHERE  c.customer_id IN ( 3, 11 );
```



Querying Vitess Cluster - Routing

```
Shard_4000_8000 (ssh) | Shard_8000_c000 (ssh)
bash-4.2$ tailf -0 /vtdataroot/tabletdata/data/sakildb-customer-4000-8000-replica-0.log | grep payment
[]

bash-4.2$ tailf -0 /vtdataroot/tabletdata/data/sakildb-customer-8000-c000-replica-0.log | grep payment
[]

VTGATE (ssh)
MySQL [(none)]>
```



Querying Vitess Cluster – @replica,@rdonly

```
4000-800 MASTER (ssh) 4000-8000 REPLICA (ssh) 4000-8000 RDONLY (ssh)
bash-4.2$ tailf /vtdataroot/tabletdata/data/sakildb-customer-4000-8000-replica-0
.log|grep payment
[]

bash-4.2$ tailf /vtdataroot/tabletdata/data/sakildb-customer-4000-8000-replica-1.log|grep p
ayment
[]

bash-4.2$ tailf /vtdataroot/tabletdata/data/sakildb-customer-4000-8000-rdonly-0.log|grep payme
nt
[]

VTGATE (ssh)
root@node1:~/sakildb#
```



Querying Vitess Cluster – Cross Shard Join

```
4000-800 MASTER (ssh)
bash-4.2$ tailf -0 /vtdataroot/tabletdata/data/sakiladb-customer-4000-8000-replica-0.log|grep payment
[]

SAKILA/0 (ssh)
bash-4.2$ tailf /vtdataroot/tabletdata/data/sakiladb-sakila-0-replica-0.log|grep staff
[]

VTGATE (ssh)
MySQL [(none)]>
```



Querying Vitess Cluster – Cross Shard Join

```
SELECT p.payment_id, 1
       p.payment_date,
       s.first_name,
       s.last_name
FROM   payment p
       JOIN staff s
       ON p.staff_id = s.staff_id
WHERE  p.customer_id = 3;
```

=

```
SELECT p.payment_id, 2
       p.payment_date,
       p.staff_id
FROM   payment p
WHERE  customer_id = 3;
```

+

N *

```
SELECT s.first_name, 3
       s.last_name
FROM   staff s
WHERE  staff_id = ?
```

N = row count of 2. query's result



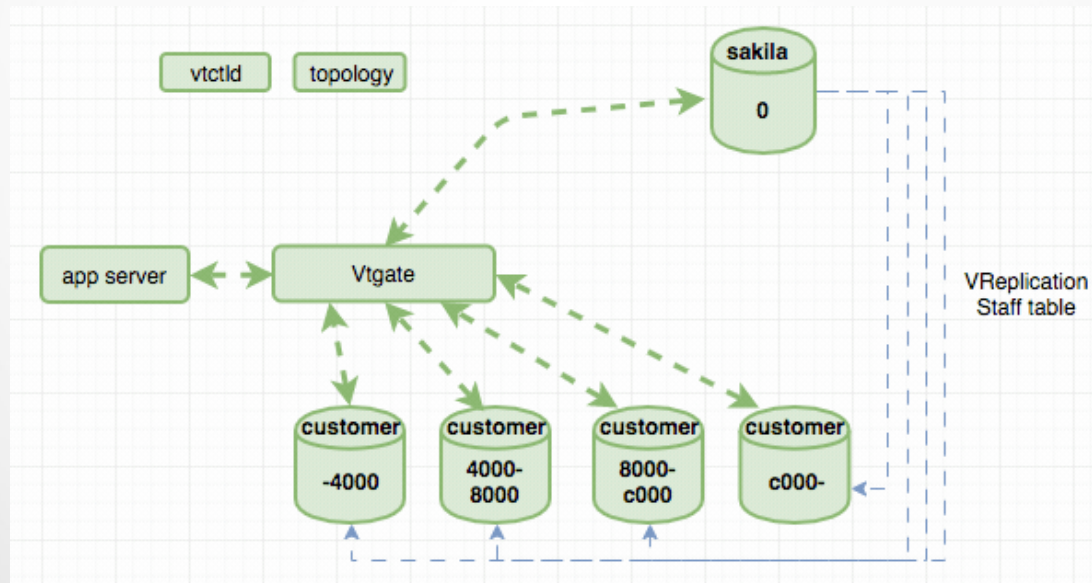
VReplication

Files:copy_schema_shard.sh,vreplication.sh,apply_routing_rules.sh,add_reference_table.sh

```
CopySchemaShard -tables staff sakila/0 customer/-4000
```

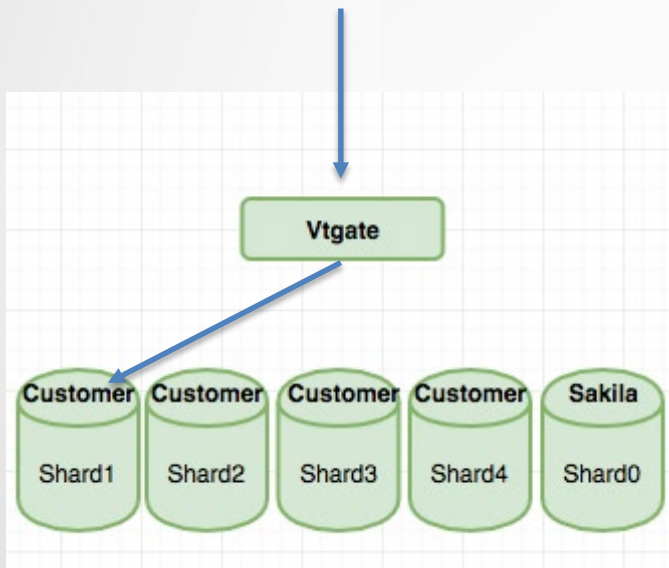
```
VReplicationExec sakiladb-0447622400 'insert into _vt.vreplication
```

```
ApplyRoutingRules -rules='{ "rules": [{"fromTable": "staff", "toTables": ["sakila.staff", "customer.staff"]} ] }'
```

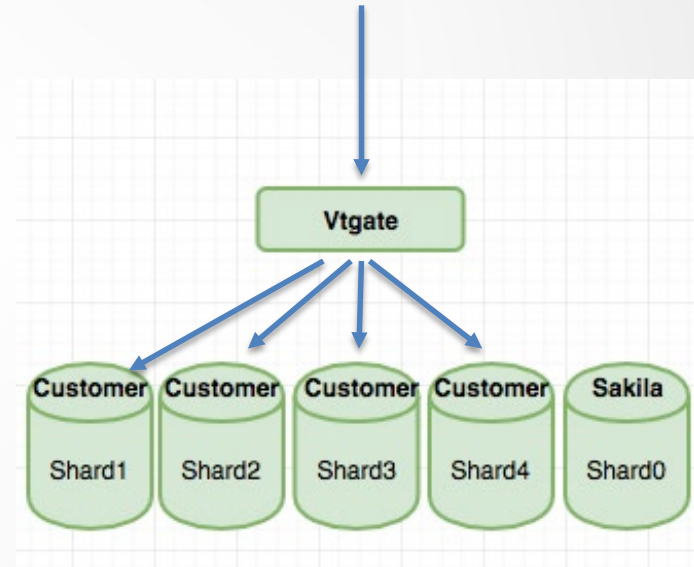


Querying Vitess Cluster – Scatter Query

```
SELECT *  
FROM payment  
WHERE customer_id = 3
```



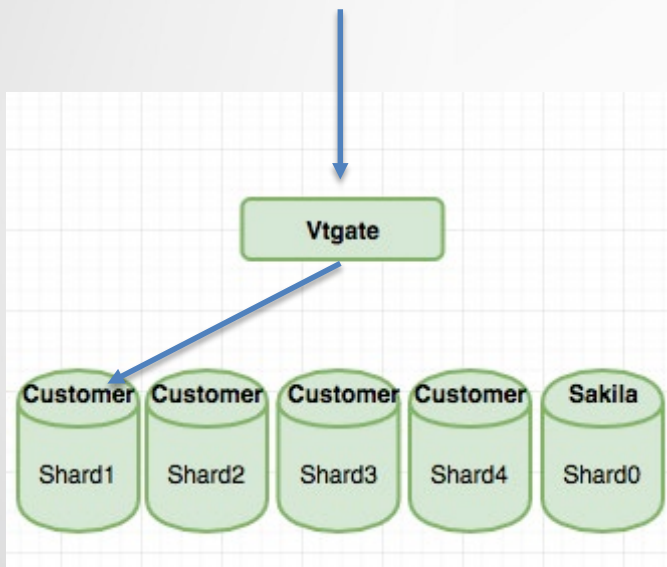
```
SELECT *  
FROM payment  
WHERE payment_id = 5
```



Querying Vitess Cluster – Scatter Query

```
CreateLookupVindex -workflow=mname -on=
```

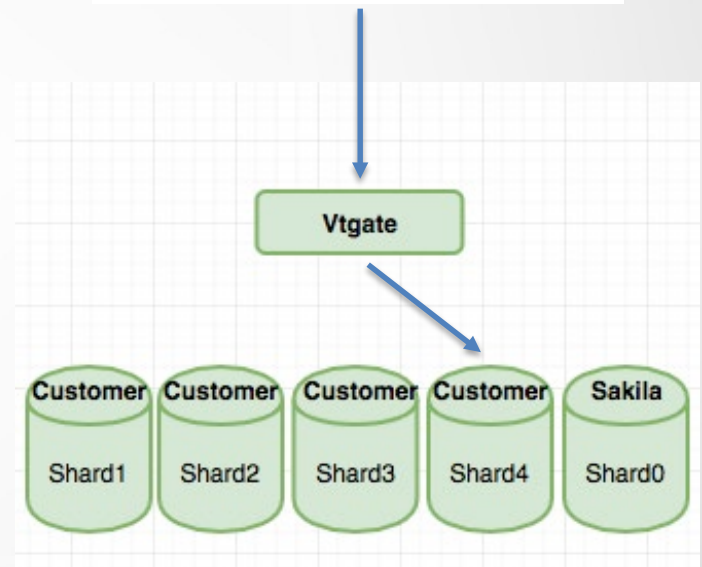
```
SELECT *  
FROM payment  
WHERE customer_id = 3
```



LookupVindex

payment_id	customer_id
1	2
2	2
3	4
.	.
.	.
.	.
.	.
.	.

```
SELECT *  
FROM payment  
WHERE payment_id = 5
```



Drawbacks: Application Transactions

Application
begin;
update payment set staff_id=2 where payment_id=5;
update rental set staff_id=2 where rental_id=1476;
commit;

Vitess Shard 1	Vitess Shard 2
begin; update payment set staff_id=2 where payment_id=5;	
	begin; update rental set staff_id=2 where rental_id=1476;
commit;	commit;

- Best Effort Commit -> Consistency Problems
- 2 Phase Commit -> Performance Cost



Drawbacks: Distributed Deadlocks

Shard Level Deadlock

Application Transaction 1	Application Transaction 2
<code>begin; update payment set staff_id=2 where payment_id=5;</code>	
	<code>begin; update rental set staff_id=2 where rental_id=1476;</code>
<code>update rental set staff_id=2 where rental_id=1476; commit;</code>	
	<code>update payment set staff_id=2 where payment_id=5; commit;</code>



Drawbacks: Distributed Deadlocks

Database Level Deadlock

Transaction 1	Transaction 2
begin;	begin;
update payment set staff_id=2 where payment_id=5;	update rental set staff_id=2 where rental_id=1476;
update rental set staff_id=2 where rental_id=1476;	update payment set staff_id=2 where payment_id=5;
commit;	commit;



Application Transactions

Application Transaction 1	Application Transaction 2
<pre>begin; update payment set staff_id=2 where payment_id=5; commit;</pre>	
	<pre>begin; update rental set staff_id=2 where rental_id=1476; commit;</pre>
<pre>begin; update rental set staff_id=2 where rental_id=1476; commit;</pre>	
	<pre>begin; update payment set staff_id=2 where payment_id=5; commit;</pre>



Q&A



vites.io



vites.slack.com



github.com/vitessio/vites





Thank You