Horizontal Sharding with Vitess

Andres Taylor, Rohit Nayak

PlanetScale

Vitess

@vitessio

- Why Shard?
- What is Vitess?
- Specifying sharding strategies
- Data sharding
- Query Planning

Why Shard?

- Physical limitations:
  - Large database size
  - Large number of rows in a table
  - High QPS / CPU /IO usage requiring high-end hardware
- Massively scalable
- On-demand scaling (up or down)
- More resilient
- Enables the use of commodity hardware
- Isolation of tenants
- Differential SLA for some tenants

Horizontal Sharding

- aka Data sharding
- Common schema on all shards
- Tables spread across databases
- Related rows on the same shard
- Challenges
  - Cross-shard queries
  - Foreign Keys
  - Unique Keys
  - Autoincrement

- Why Shard?
- What is Vitess?
- Specifying sharding strategies
- Data sharding
- Query Planning

Vitess is a scalable, distributed database system built around MySQL

Vitess

CLOUD NATIVE COMPUTING FOUNDATION

GRADUATED PROJECT

@vitessio

What is Vitess?

Cloud Native Database

Massively Scalable

Highly Available

MySQL Compatible

Works With

Database Frameworks

ORMs

Legacy Code

Third-Party Applications

Logical Database

Many Physical Databases

Query Routing

gRPC Clients MySQL protocol

Single Connection

slack

New Relic.

Square

Flipkart

HubSpot

peak

Pinterest

shopify

nozzle

weave

GitHub

JD.COM 京东

Quiz of Kings

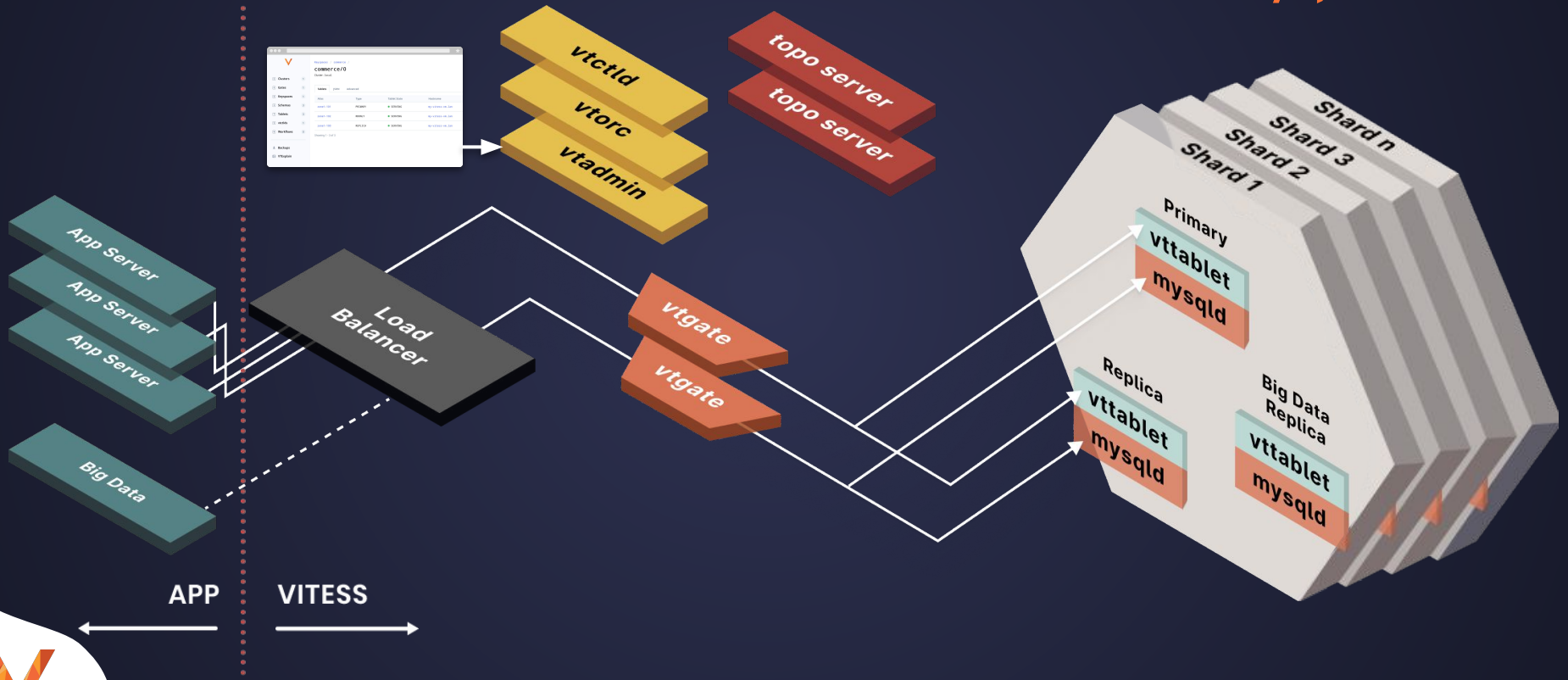
stitchlabs

PlanetScale

Vitess

@vitessio

Architecture



Vitess

@vitessio

- Why Shard?
- What is Vitess?
- Specifying sharding strategies
- Data sharding
- Query Planning

Vitess

@vitessio

Sharding In Vitess

- Vertical Sharding:
  - Multiple unsharded keyspaces, related tables split across keyspaces
  - Use **`MoveTables`** VReplication workflows
  - Intermediate step before data sharding

- Horizontal Sharding:
  - Sharded Keyspace: defined by a VSchema
  - Sharding Key: per table, one or more columns,
  - Primary Vindex: maps sharding key to shard
  - Secondary Vindexes: for common predicate columns
  - Use **`Reshard`** VReplication workflows
  - Use Sequences for Autoincrements, backed by unsharded keyspace
  - Reference and **`Materialize`**'d tables for data locality

Vitess

@vitessio

Sharding Strategies

- Range-based Sharding
  - {-}, {-80,80-}, {-80, 80-c0, c0-dc00, dc00-dc80, dc80-}
  - row => 64 bit keyspace_id, using one or more column values
  - Mapping done by a Vindex function
  - One shard per key range of contiguous keyspace_ids
- Sharding Key: per-row tuple of one or more column values
- Primary Vindex: projects the sharding key to a keyspace id (and hence shard)
- Vindexes defined in a VSchema
- Vindex types: binary, xxhash, custom json map, unicode_loose_xxhash, multicol
- Generic: strategy is not hard coded, nor is the app sharding aware
- Sharding key can be changed using `MoveTables` workflows

- Why Shard?
- What is Vitess?
- Specifying sharding strategies
- Data sharding
- Query Planning

Performing Reshards

- Sharding
  - `Reshard -w wf1 --target-keyspace customer Create --source-shards '0' --target-shards '-80,80-'`
- Resharding
  - `Reshard -w wf2 --target-keyspace customer Create --source-shards '-80' --target-shards '80-c0,c0-'`
- Control plane cli: `vtctldclient`
  - `Create` → `SwitchTraffic` [→ `ReverseTraffic`] → `Complete`
  - `Show` / `Progress` to debug/monitor

Vitess

@vitessio

VReplication Workflows

- Target streams from source vttablets (replica/primary)
- Starts with a Copy phase
  - One table at a time, in batches
  - On Source: Take consistent snapshot, streaming select
  - On Target: Bulk insert into target
  - State maintained in a sidecar database.
  - Between tables/batches, stream binary logs, with dmls for copied ranges
- Move to Running (binlog streaming) phase until cutover

Vitess

@vitessio

VReplication Workflows

- Fast, eventually consistent
- Near-zero downtime cutover
- Resumable, resilient to:
  - primary failovers,
  - network outage
- Throttling, based on:
  - replica lag
  - history list length
  - custom mysql query: max #connections, #threads_running,

Indicative Performance

- One Table: 170GB, 3.2B rows, 3 secondary indexes
- Copy: 17K rows/s, 13 hours + 4 hours reindex, (42 hours wo reindex)

- One Table: 4.15TB, 7.8B rows, 3 secondary indexes
- Copy: 62K rows/s, 35 hours total, 1=>4 shards

- Performance factors
  - Environment: CPU/IO/Memory, Network latency/bandwidth, MySQL settings
  - Application:
    - #tables, #rows, row widths, data types/blob, PK types, Indexes
    - write/read QPS, large transactions
  - VReplication Settings: Packet Size, Copy phase duration, Parallel copy, Throttling

Sharding Stories

- Scaling Datastores At Slack With Vitess
  https://slack.engineering/scaling-datastores-at-slack-with-vitess/
- Sharding Cash
  https://developer.squareup.com/blog/sharding-cash/
- Horizontally Scaling The Rails Backend Of Shop App With Vitess
  https://shopify.engineering/horizontally-scaling-the-rails-backend-of-shop-app-with-vitess
- Scaling Etsy Payments With Vitess
  https://www.etsy.com/codeascraft/scaling-etsy-payments-with-vitess-part-1--the-data-model
- One Million Queries Per Second With MySQL
  https://planetscale.com/blog/one-million-queries-per-second-with-mysql
- Vinted Vitess Voyage: Chapter 3 - The Great Migration
  https://vinted.engineering/2023/04/27/vinted-vitess-voyage-chapter-3-the-great-migration/

- Why Shard?
- What is Vitess?
- Specifying sharding strategies
- Data sharding
- Query Planning

Meet the vtgate Query Planner

(all the following images by DALL-E, except one)

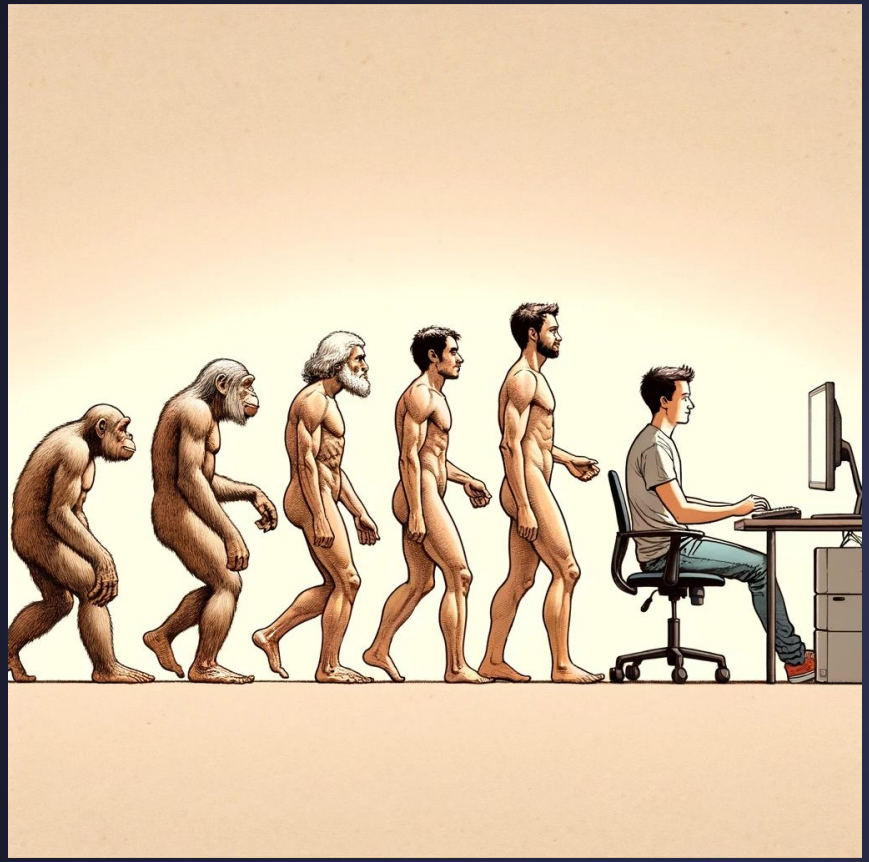
Vitess

@vitessio

Beyond Naive Approach

Vitess

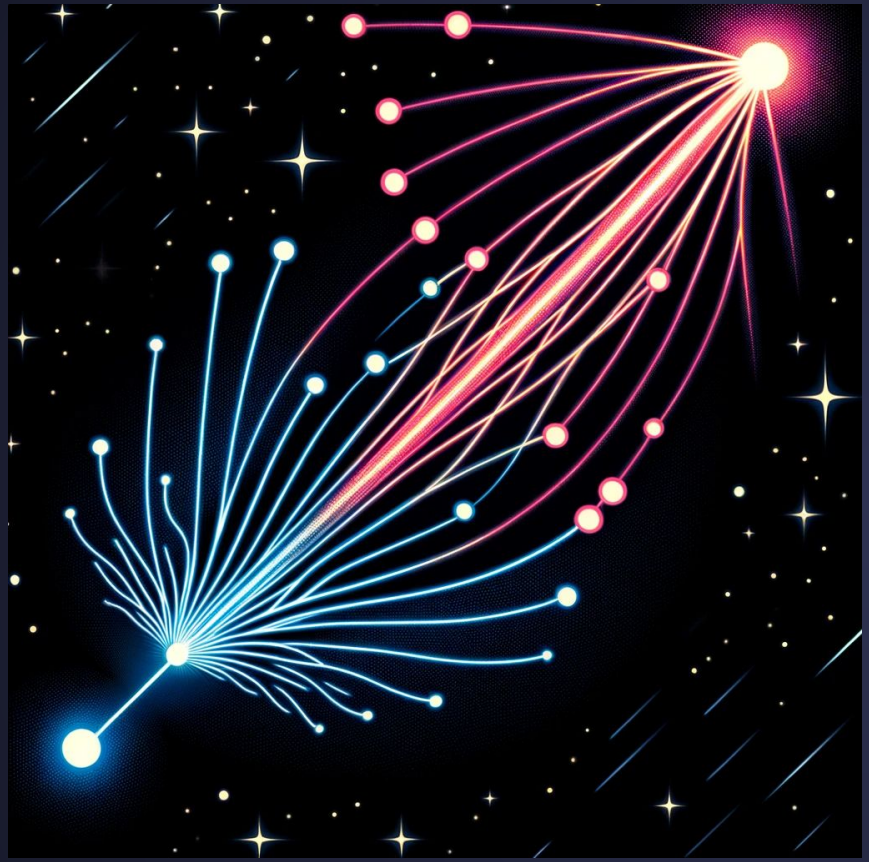
@vitessio



Evolution of the Vitess Planner

Vitess

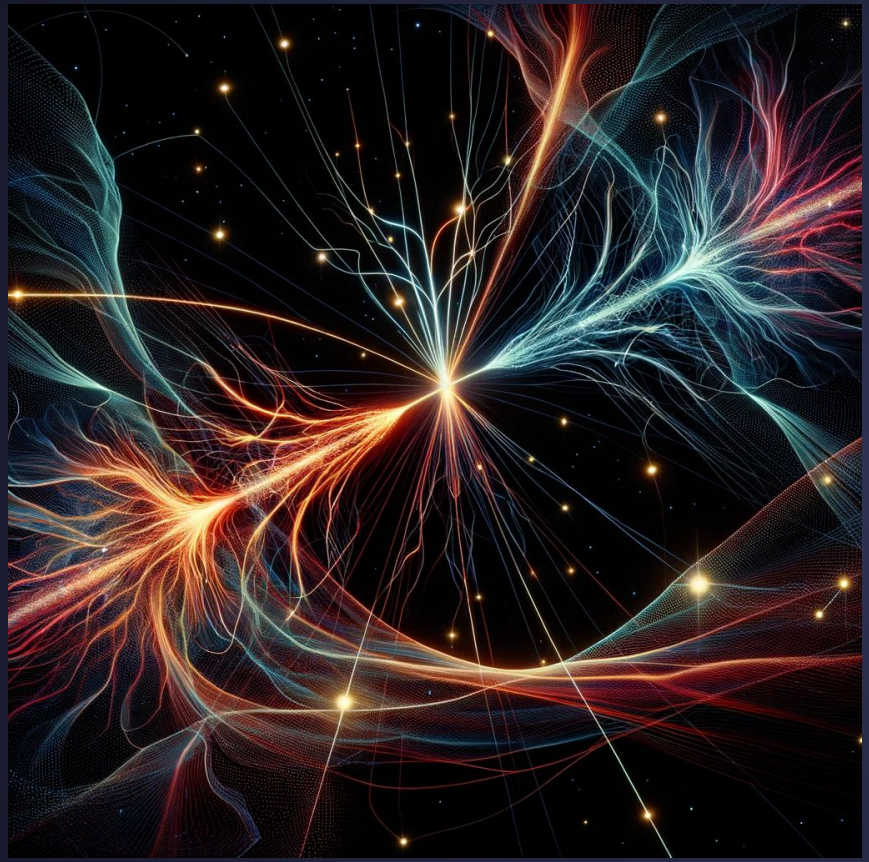
@vitessio



The v3 Planner Breakthrough

Vitess

@vitessio



Gen4 Planner: A New Era

Vitess

@vitessio

First Steps in Query Planning

Parsing
string -> AST

Semantic Analysis
AST -> AST++

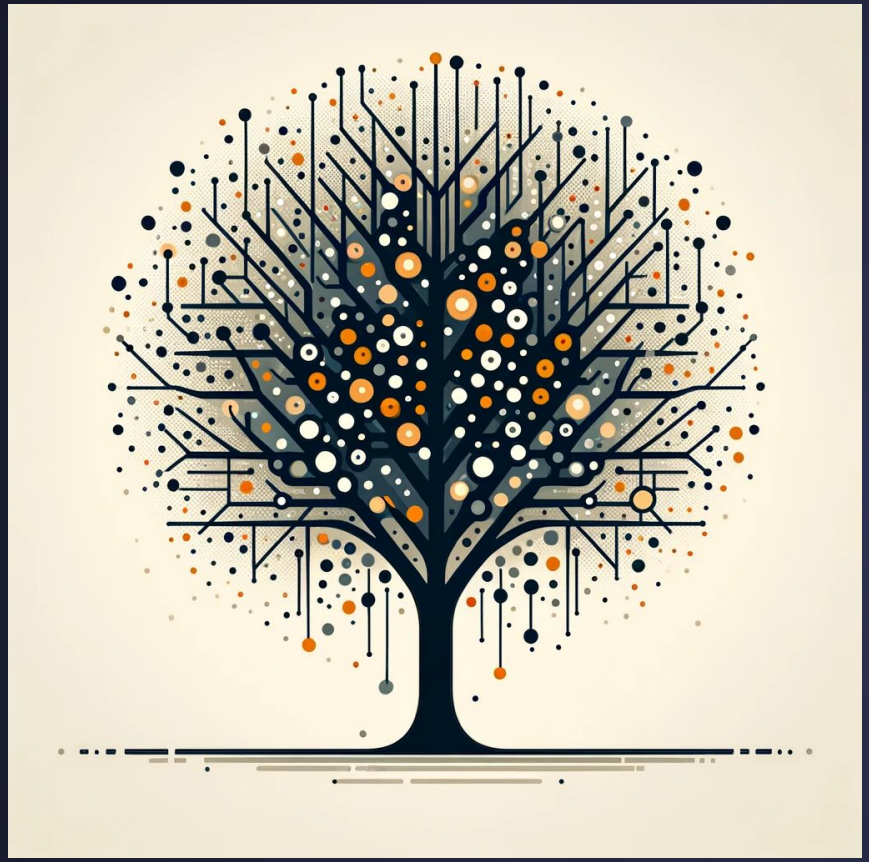
Vitess

@vitessio

Simplifying Unsharded Queries

Vitess

@vitessio



From AST to Operator Tree

Vitess

@vitessio

The Route Operator in Action

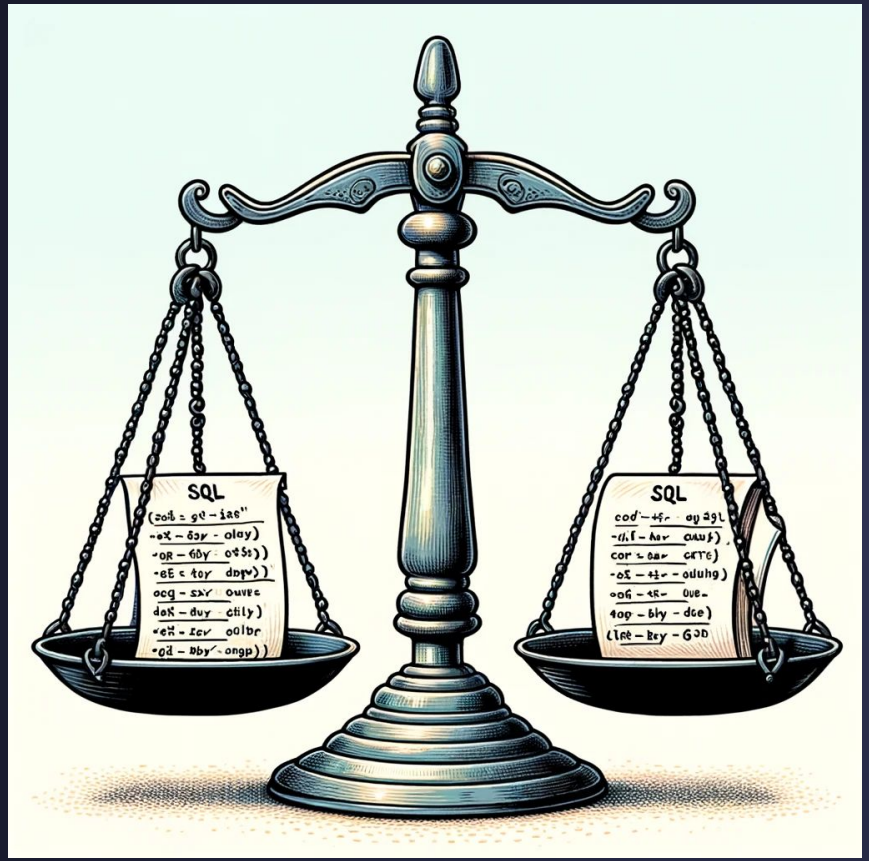
Vitess

@vitessio

Understanding Vindexes in Sharding

Vitess

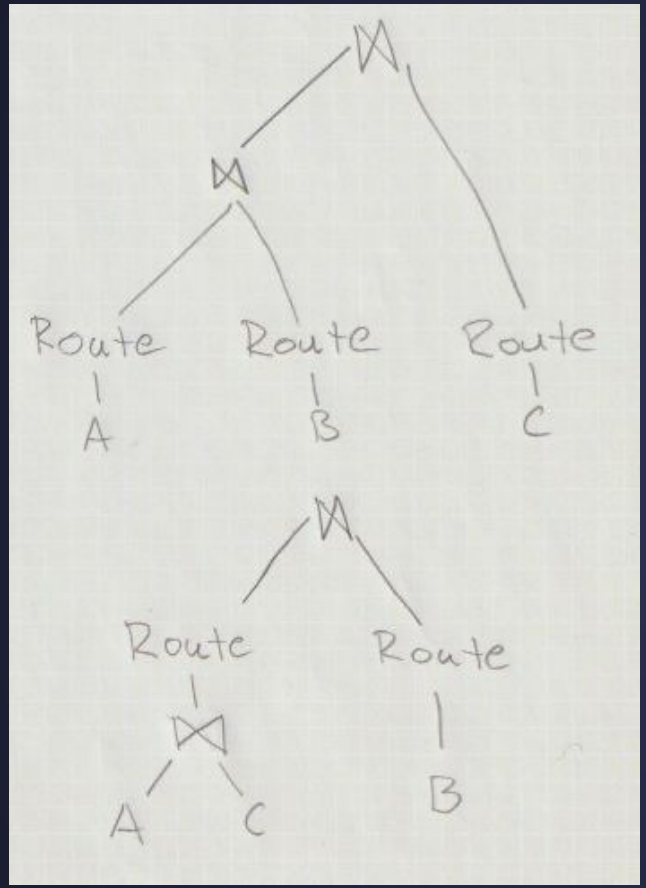
@vitessio



Cost Estimation

Vitess

@vitessio



Optimizing Joins in Query Planning

Vitess

@vitessio

Tree Rewriting

Vitess

@vitessio

```
SELECT count(*)
FROM user u
   JOIN user_extra ue
   ON u.id = ue.foo
```

Phases of Query Planning

Vitess

@vitessio

Initial tree

```
Horizon
└── QueryGraph (`user`, user_extra)
```

PHASE: physical transformation

```
Horizon
└── ApplyJoin (ue.foo cols: )
    ├── Route (Scatter:user)
    │   └── Table (user_extra AS ue)
    └── Route (Unique user[user_vindex|:ue_foo])
        └── Filter (u.id = :ue_foo)
            └── Table (user AS u)
```

Vitess

@vitessio

PHASE: horizon expansion

```
Aggregator (count(*))
└── ApplyJoin
    ├── Route
    │   └── Table
    └── Route
        └── Filter
            └── Table
```

Vitess

@vitessio

PHASE: split aggregation and push under join

```
Aggregator (sum_count(c3) AS count(*))
└── Projection (c1 * c2 AS c3)
    └── ApplyJoin (cols: c1, c2)
        ├── Aggregator (count(*) c1 GB ue.foo)
        │   └── Route
        │       └── Table
        └── Aggregator (count(*) as c2)
            └── Route
                └── Filter
                    └── Table
```

Vitess

@vitessio

>>>>>>>> push aggregation under route

```
Aggregator
└── Projection
    └── ApplyJoin
        ├── Route
        │   └── Aggregator(count(*) as c1 GB ue.foo)
        │       └── Table
        └── Route
            └── Aggregator (count(*) as c2)
                └── Filter
                    └── Table
```

Vitess

@vitessio

After offset planning

```
Aggregator (sum_count(0))
└── Projection (:0 * :1)
    └── ApplyJoin
        ├── Route (Scatter:user)
        │   └── select count(*), ue.foo
        │       from user_extra as ue
        │       group by ue.foo
        └── Route (Unique user[user_vindex|:ue_foo])
            └── select count(*)
                from `user` as u
                where u.id = :ue_foo
```

Vitess

@vitessio

Creating the Illusion

Vitess

@vitessio

The Future: gen5 and Cardinality Model

Vitess

@vitessio

Questions and Discussions

Vitess

@vitessio