Physical Data modeling with YugabyteDB

Speaker Introduction

Bryan Whitmore

Principal Pre-Sales Engineer

YugabyteDB Basic Data modeling

- The process
- Some general rules of thumb
- Yugabyte specifics
- Some Examples

The process

- Preparation
  - Start with your existing database design
  - What are your common Select and DML patterns
  - Where does performance matter
  - What are your specific objectives
- Steps
  - Review primary keys
  - Review Indexes
  - Test queries and Look at query plans
  - Consider denormalization/refactoring tables

Some Basic background

- YugabyteDB is a distributed database
  - Data is on multiple nodes and performance will suffer if the data for your query is spread over all of them. Plus you have killed scaling.
- YugabyteDB distributes data by primary key
  - This makes primary key selection critical.
  - Using an ORM created synthetic key while common with other databases is going to ensure poor performance
  - Indexes follow the same rules as primary keys

The Three rules

1. Minimize the number of I/Os each request to the database takes
   a. Both Storage and Network but most especially Network
2. Maximize the number of the database nodes that will take traffic for all requests
   a. This may seem to be in direct contradiction to 1. and it sometimes is but it will help to spread out the workload and avoid hotspots.
3. Keep your use cases in mind

Our tools to implement the three rules

1. Primary Keys - All tables must have one - the choice is critical
2. Secondary Indexes - Use with care but - they can be very useful
3. Minimize the number of tables/tablets
4. Denormalize

Primary Key Selection

- The primary key is used to place table data into tablets distributed over the database cluster.
- The primary key has two components
  - Partition - the column(s) which are hashed to pick a tablet
  - Cluster - the remaining columns which are used to define storage order within the partition
- A sequence value by itself is almost never a good choice for a primary key
- A Primary key with a Sequence embedded is also not a very good choice
  - Consider if a UUID meets the needs instead

Yugabyte Primary Key by Store

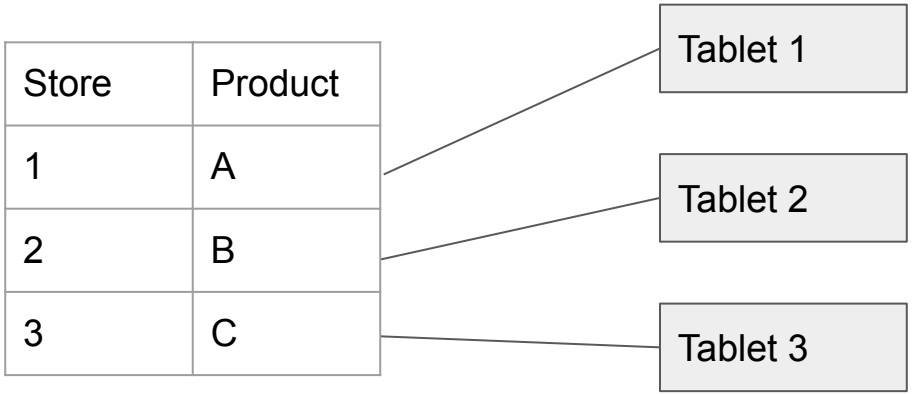
```
create table inventory (
    Store_id        text not null,
    Product_id      text not null,
    data            jsonb,
    meta            jsonb,
    primary key ((Store_id) hash, Product_id)
);
-- Store_id hashes to a specific tablet/shard
-- All rows with different product_ids but the same store_id will be on
the same tablet
```

Yugabyte Primary Key by Product

```
create table inventory (
    Store_id        text not null,
    Product_id      text not null,
    data            jsonb,
    meta            jsonb,
    primary key ((Product_id) hash, store_id)
);
-- Product_id hashes to a specific tablet/shard
-- All rows with different store_id but the same product_id will be on
the same tablet
```

Spread and Condense

- Spread workload across all tablets/shards helps to scale writes
- Collect related data on tablets to minimize reads



Yugabyte Alternate key options

```
create table inventory (
    Store_id        text not null,
    Product_id      text not null,
    Instore_count   integer not null,
    data            jsonb,
    meta            jsonb,
    primary key ((Stor_id) hash, Product_id)
);
create unique index product_id_idx on inventory (Product_id hash,
store_id);
create index product_inventory_idx on inventory (Product_id hash,
Instore_count);
Create index product_inventory_full_idx on inventory (product_id hash,
Instore_count) include (data, meta);
```

Yugabyte indexes

```
create unique index product_id_idx on inventory (Product_id hash,
store_id);
-- creates a secondary index table based on product_id and store_id
create index product_instore_count_idx on inventory (Product_id hash,
Instore_count);
-- Creates a secondary index on product and instore count good for
-- searches on store inventory levels
Create index product_inventory_full_idx on inventory (product_id hash,
Instore_count) include (data, meta);
/* same as above but includes all the data in the table, reduces I/O on
   retrieving all the data from a row without having to go back to the
   base table */
```

Index Summary

- Make your primary key a compound key
  - which Yugabyte can use to pick a tablet for you to store clusters of data.
  - If your table already has a unique compound key it is going to be your best candidate
- Take advantage of covering indexes
  - Helps to increase the number of index only scans
- Avoid Sequences where you can
  - Definitely bad for the partition portion of the primary key.
  - Consider a UUID instead
  - Sequences are maintained in the PostgreSQL catalog tablet
    - Too much activity against that tablet can become a hot spot
    - If you must use sequences consider using a large cache to reduce Catalog references
    - Don’t use serial data type which uses cache of 1

Denormalization

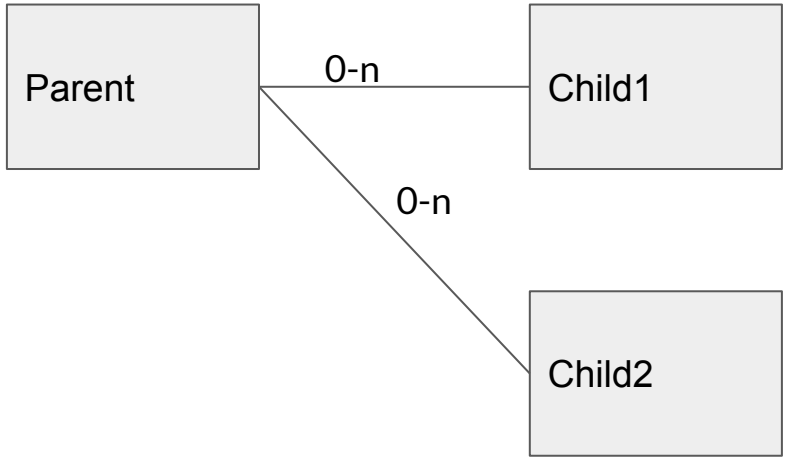
- Why denormalize
    - Helps to reduce the number of tablets touched in a query
    - Concentrates data close together
        - Rows from different tables are pulled together in one table
    - Can help to return data more closely aligned with the applications needs.

- Why not denormalize
    - My Query times are already good enough!
    - Duplicate data always happens when you denormalize
        - Sometimes yes, But often that is not true
    - Its harder to do joins.
        - Sometimes true. More often, you no longer need joins.

How to denormalize

- Works best in hierarchical models
- Some Common techniques
  a. Pull 0-n relationships with small n into complex objects
  b. Pull 0-n relationships into the primary table as row types
- Replace small reference tables with enums

Denormalization



Denormalize by object

```
Create table denorm1 (
    Id          uuid not null,
    Cid         text,
    Data        jsonb,
    Child1      jsonb,
    Child2      jsonb,
    Meta        jsonb,
    Primary key (cid, id);
```

Denormalize by row

```
Create table denorm2 (
    Id              uuid not null,
    Row_type        enum,
    Child_id        uuid not null,
    Cid             text,
    Data            jsonb, -- contains different data based on
    Meta            jsonb, -- row type
    Primary key (cid, child_id, id, row_type id);
```

Summary

- It is all about I/O
- Minimize I/Os for a single query - Minimize response time
- Maximize data placement on to as many tablets as you can - Maximize throughput
- Its all about index choices (primary key and secondary indexes)
- And, where necessary denormalization. - Fewer tables and fewer I/Os

Physical Data modeling with YugabyteDB