Data Bootcamp

Week 1:

Day 1: Dimensional Data Modelling & Complex Data types

Dimensions – attributes of an entity

- Come in 2 types: ***fixed(e.g. birthdate)*** and ***Slowly changing*** (e.g favorite food in your entire life i.e. time dependent)

OLTP (low-latency, low volume queries)

OLAP (optimizes for large volume, group by queries, minimizes JOINs)

Master Data (sits in between OLTP and OLAP, optimizes for completeness of entity definitions, deduped)

The complete data cycle (layers of data modelling):

Production Data snapshots → Master Data → OLAP Cubes (slice and dice) → Metrics

Master Data is like combining all transactional datasets/tables to create this master data set (deduped data).

OLAP cubes are like the data is flattened and it contains like multiple entries for an entity.

Metrics are like the aggregations done on an OLAP cube resulting in a single number e.g. average listing price for a region.

Cumulative Table Design:

Implies keeping history because that's how you will calculate cumulative. E.g State transition tracking for a user to track their activity state on any online platform.

Strengths:

- ➔ Easy trend analysis
- ➔ Historical analysis without shuffle.

Weakness:

- ➔ Can only be backfilled sequentially.
- ➔ Handling PII data can be painful

Struct vs Array vs Map

-> In Struct: keys rigidly defined, good compression, values can be of any data type.

-> In Map: keys are loosely defined, ok compression and values have to be of the same data type.

-> In Array: Ordinal data; list of values have to be of the same data type (but you can have map and structs as a data type)

Temporal Cardinality Explosions of Dimensions

When you add a temporal (time) aspect to your dimensions and the cardinality increases by atleast 1 order of magnitude.

Run length encoding compression

Quite important compression technique in Big data which makes parquet file formats so successful. But shuffle in spark can ruin this (since shuffle happens in spark when you apply group by or join). Post shuffle the rows get sorted because of which the run length encoding compression can reduce and not include the number of rows as we expect.

Two ways to handle it:

- ➔ Re-sort the data again after grouping before applying run-length compression. (not so efficient)
- ➔ Make use of complex data types like struct, array etc and then explode then post shuffle for applying run length compression (keeps the temporal pieces together).

This way you can make sure that the downstream data does not explode and works as per the expectation.

Day 2: Building Slowly Changing Dimensions

Slowly Changing Dimensions

- They come with a time frame (i.e. tend to change with time).

Idempotency – The ability for your pipeline to produce same results irrespective of where it's running (important property of data pipelines that adds to data quality).

Non-idempotent – Does not produce same data when it runs (kind of silent failure, data inconsistencies).

What can make a pipeline non-idempotent:

- Insert into without truncate (***solution: use merge or insert overwrite***)
- Using '**start_date >'** without specifying a corresponding **'end_date <'** (can also result in OOM exceptions)
- Not using a full set of partition sensors (i.e. pipeline might when there is no/partial data).
- Not using **depends_on_past** for cumulative pipelines (it needs sequential processing and you cannot do parallel processing).

Point to Note: Production and Backfill failures are the same.

- Relying on the '**latest'** partition of a not properly modeled SCD table (Cumulative table design AMPLIFIES this bug).
- Relying on the **'latest'** partition of anything else.
- However, in **case of Backfilling you can rely on latest data** provided you have a properly data modeled SCD table.

Note: Using SCDs in data modelling is great if the dimension changes really slowly because that would help you to get great deal of data compression.

How to model dimensions that changes:

- Latest snapshot (can lead to issues in case of backfilling)
- Daily Partitioned snapshots.
- SCD types 1,2,3

SCD Types:

Type 0: Does not change, no temporal component (e.g birth date)

Type 1: You only care about the latest value (has the tendency to make your pipelines non-idempotent, is not a good solution for analytics).

Type 2: You care about the value from 'start_date' to 'end_date' (they are definitely idempotent). There will be more than 1 row per dimension.

Type 3: You care about 'original' and 'current' value. You only have 1 row per dimension.

(Zach's quote: As a data engineer you have to know that not every pipeline you build has to be a Ferrari.)

Day 3: Graph Data Modelling

- Schema is flexible

Additive vs Non-Additive Dimensions:

Additive Dimension – which you cannot double count (for eg – population = 20 year olds + 30 year olds + 40 year olds ……)

Non-additive = no. of honda drivers =/ no. of corolla drivers =/ no. of Hyundai drivers

How Additivity helps: you do not need to use count distinct on preaggregated dimensions.

(Only applies when you are using COUNT() or Ratio metric and not on other aggregations like SUM())

General Thumb Rule: Can a user belong to the two groups at the same time (age is a good example for this use case).

No. of users on your app = users on iphone + users on android

Enums:

- Are good for low-medium cardinality (probably less than 50 values).
- Country is an example where enums start to struggle.

Why use Enums:

- Built in data quality
- Built in static fields
- Built in documentation

Enumerations and subpartitions

- Enums make an amazing subpartitions because
    - You get an exhaustive list.
    - They chunk up the big data problem into manageable pieces.

Model data from disparate sources into a shared schema (Flexible schema) – map data type is commonly used for this.

Drawbacks for flexible schema:

- Compression is painful (especially if you use json and maps).
- Readability, querability.

How graph data modelling is different:

- It is relationship focused and not entity focused (edges and nodes). Each edge takes on two nodes.
- Usually the model looks like (Entity Modelling):
    - Identifier: STRING
    - Type: STRING
    - Properties: MAP<STRING, STRING>
- Relationship modelling is a bit more in depth
    - Subject_identifier: STRING
    - Subject_type: VERTEX_TYPE
    - Object_identifier: STRING
    - Object_type: VERTEX_TYPE
    - Edge_type: EDGE_TYPE
    - Properties: MAP<STRING, STRING>

Week 2:

Day 1:

Fact Data Modelling

- Fact is something that happened, something that is true.
- Facts are not slowly changing.
- Volume of fact data is 10-100x the volume of dimension data.
- The fact volume primarily depends upon how many actions were taken per dimension.
- Fact data can need a lot of contexts for effective analysis.
- Duplicates in facts are quite common than dimensions.

Normalization vs Denormalization

- Normalized facts do not have any dimensional attributes, just IDs to join to get the information (increased data integrity).
- Denormalized facts contain dimensional attributes for quicker analysis at the cost of more storage.

Note: Smaller the scale, normalization is the solution

- Raw logs and fact data are quite linked and cleaner logging can actually help in higher fact data quality.
- Build raw logs into highly trusted fact data.
- Facts can be 'who', 'what', 'where', 'when' and 'how'; each corresponding to an ID like user_id, device_id, event_date, action etc.
- Prefer client-side logging instead of server-side logging (go for UTC timezone to avoid confusions).

Options when working with High Volume Fact Data

- Sampling (works best for metric-driven use-cases where imprecision is not an issue).
- Bucketing (on the basis of some important dimensions). Bucket joins can be much faster than shuffle joins.
- Sorted-merge buckets (SMB) joins can do joins without Shuffle at all.

Hourly deduping with Daily Microbatch (Helps when working with billions of records everyday).

Day 2:

- You can aggregate facts and turn them into dimensions.
- Bucketize aggregated fact values proves useful in reducing the cardinality.

Dimensions Vs Facts

Dimensions:

- show up in GROUP BY while doing some analytics.

- Can be high cardinality (like country) or low cardinality (like gender).
- Originates from a snapshot of a state.

Facts:

- Usually, aggregated values when doing analytics like SUM, COUNT, AVG.
- Almost always higher volume than dimensions.
- Generally, come from events and logs.

Question: Is the price of a night on Airbnb a fact or dimension?

- The host sets the price which sounds like an event.
- It can be aggregated (SUM, AVG etc) like regular facts.
- But it is a dimension since it is an attribute of the price of a night.

Categorical Fact/Dimensions

- Dimension that is derived from fact data.
- Often calculated with case when logic and bucketizing (eg – Airbnb superhost).
- Date List Data structure extremely efficient for dates_active data storage for monthly/weekly analysis (in the form of 0s and 1s).

Day 3:

Minimizing Shuffle and reducing facts:

Why should shuffle be minimized?

- Big data leverages parallelism and shuffle is a bottleneck for parallelism.

Types of queries that are parallelizable:

- Extremely Parallel: select, from, where
- Somewhat Parallel: group by, join, having
- Not Parallel: order by (not the one with the window function)

These keywords effect SHUFFLE in different ways.

Key Note: Shuffle happens when you need all the data from a specific key on a specific machine.

How to make group by more efficient?

- Assign some buckets in group by
- Reduce the data volume as much possible

How Reduced Fact Data helps?

- Fact data usually has schema: user_id, action, partition_date *(which has high volume, 1 row per event)*.

- Daily aggregate often has schema: user_id, action_count, date_partition *(Medium size volume, 1 row per user per day)*.
- Reduced fact, going one step further with schema: user_id, action_cnt_array, month_start_partition/year_start_partition *(Low volume, 1 row per user per month/year)*.
- Daily dates are stored as an offset of month_start/year_start
    - First index is for date month_start + 0 days
    - Last index is for date month_start + array_length - 1