Câu 1: Phân biệt giữa Kimball, One Big Table và Relational Modeling. Đối với vai trò là Data Engineer, bạn sẽ lựa chọn phương pháp nào trong từng trường hợp cụ thể.

Ba mô hình này là cách tổ chức dữ liệu trong một hệ thống, mỗi mô hình có ưu điểm và nhược điểm riêng. Việc lựa chọn phương pháp mô hình hóa dữ liệu phụ thuộc vào mục tiêu kinh doanh, hiệu suất truy vấn, quy mô dữ liệu và cách dữ liệu được sử dụng.

1. Kimball (Dimensional Modeling - Mô hình hóa dữ liệu chiều)

Đặc điểm:

- Dựa trên Data Warehouse với mô hình Star Schema hoặc Snowflake Schema.
- Chia dữ liệu thành Fact Table (bảng sự kiện) và Dimension Table (bảng chiều).
- Được tối ưu hóa để truy vấn báo cáo nhanh chóng và hỗ trợ BI (Business Intelligence).
- Ít quan hệ phức tạp, dễ hiểu với người dùng phân tích dữ liệu.

2. One Big Table (OBT - Mô hình bảng phẳng)

Đặc điểm:

- Dữ liệu được gộp vào một bảng duy nhất, không có bảng quan hệ.
- Truy vấn đơn giản do không cần JOIN, phù hợp với hệ thống BigQuery, Snowflake, hoặc Data Lake.
- Dữ liệu lặp lại nhiều lần, có thể gây lãng phí lưu trữ.
- Dễ sử dụng với các công cụ ML/AI vì dữ liệu đã được chuẩn hóa thành một bảng duy nhất.

3. Relational Modeling (Mô hình quan hệ - 3NF)

Đặc điểm:

- Chuẩn hóa dữ liệu theo 3NF (Third Normal Form), chia nhỏ thành nhiều bảng quan hệ để giảm trùng lặp và tối ưu cập nhật dữ liệu.
- Thường được sử dụng trong hệ thống OLTP (Online Transaction Processing), nơi dữ liệu được cập nhật liên tục.
- Cần nhiều phép JOIN, có thể gây ảnh hưởng hiệu suất truy vấn khi dữ liệu lớn.

=> Theo em, nên chọn Kimball để làm Data Warehouse, nơi lưu trữ dữ liệu lịch sử, thích hợp cho việc phân tích và báo cáo, không nên chọn Kimball để thiết kế cho Database nơi giao tiếp trực tiếp với User vì Kimball tối ưu cho truy vấn đọc lớn (OLAP - Online Analytical Processing), không phải cho việc cập nhật dữ liệu liên tục.

One Big Table có thể dành cho Data Lake hoặc dữ liệu có kích thước nhỏ và không quá phức tạp. Truy vấn đơn giản, tránh việc JOIN nhiều bảng. Không cần tối ưu hóa sâu về

hiệu suất. Ứng dụng trong việc lưu trữ các raw data như log của các hệ thống game, bán hàng,...

Relational modeling thì phù hợp cho thiết kế database, nơi dữ liệu thay đổi, cập nhật liên tục, tương tác trực tiếp với user. Ví dụ như trả về số dư tài khoản sau khi giao dịch thành công. Ứng dụng trong các hệ thống quản lý đơn hàng

Câu 2: Trong Bigquery, tại sao chỉ partition được column dạng INTEGER, time-unit, ingestion time?

Bigquery chỉ partition được column dạng integer, time-unit, ingestion time vì nhưng data type này có trật tự tuyến tính(có thể sắp xếp được).Các số nguyên hoặc ngày tháng có thứ tự logic, giúp BigQuery dễ dàng xác định khoảng cần quét.
Khi truy vấn theo khoảng ví dụ between x and y, BigQuery chỉ cần quét một đoạn liên tục.
Các dạng như String không có trật tự xác định. Nếu partition theo STRING, BigQuery không thể xác định khoảng giá trị hợp lý để bỏ qua partitions, làm giảm hiệu suất truy vấn.

Câu 3: Trong Bigquery, mô tả nguyên lý hoạt động (ở cấp độ engine) của clustered table, trong trường hợp sử dụng column dạng STRING.

Cách hoạt động của Clustering (ở cấp độ Engine)

1. Chia nhỏ dữ liệu thành Blocks
    - Khi dữ liệu được nạp vào bảng, BigQuery tự động chia nhỏ thành blocks (~ 100MB/block).
    - Mỗi block lưu trữ một phạm vi giá trị của cột cluster.
2. Sắp xếp dữ liệu trong mỗi block
    - Dữ liệu trong mỗi block được sắp xếp theo giá trị của cột cluster.
    - Không có index cố định, nhưng việc sắp xếp giúp BigQuery tìm kiếm hiệu quả hơn.
3. Tạo metadata cho từng block
    - BigQuery lưu min/max value của cột cluster trong metadata của từng block.
    - Khi truy vấn, BigQuery chỉ đọc các blocks có giá trị phù hợp, bỏ qua phần còn lại (cluster pruning).

ví dụ 1 cột tên ‘Country’ gôm : Vietnam, America, Thailand, Korea,.... Khi áp dụ clustered cho cột này, các giá trị trong cột sẽ được chia thành các block(như cột Country ở đây thì sẽ sắp xếp theo bảng chữ cái vì là dạng string). Block 1 gồm America,Belgium,...Block 2 gồm Canada, China,..Tương tự với các Block còn lại. Khi thực hiện SELECT * FROM table WHERE country = 'France'; Bigquery Chỉ đọc Block 3 vì France nằm trong khoảng từ Canada-France

Câu 4: Trong Bigquery, so sánh hàm APPROX_COUNT_DISTINCT và COUNT DISTINCT.

1. COUNT DISTINCT

- Count Distinct thông thường sẽ đọc toàn bộ dữ liệu, nó sẽ duyệt qua tất cả các dòng, lưu trữ các giá trị duy nhất vào 1 tập hợp sau đó đếm số phần tử trong tập hợp này.
- BigQuery sử dụng **distributed computing (tính toán phân tán)** để tăng tốc độ:

1. Chia dữ liệu thành nhiều node xử lý song song.
2. Mỗi node sẽ tính số lượng giá trị duy nhất trong phần dữ liệu của nó.
3. Gộp kết quả từ các node lại để có danh sách giá trị duy nhất cuối cùng.
4. Đếm số phần tử trong danh sách hợp nhất này.

2. APPROX_COUNT_DISTINCT

- BigQuery sử dụng **thuật toán HyperLogLog++ (HLL++)** để ước lượng số lượng giá trị duy nhất trong tập dữ liệu một cách nhanh chóng mà chỉ sử dụng bộ nhớ nhỏ.

Hashing

- Thay vì lưu toàn bộ giá trị duy nhất, HyperLogLog++ **băm (hash)** mỗi giá trị thành một chuỗi bit cố định.
- Điều này giúp chuyển các giá trị bất kỳ (string, số,...) thành một biểu diễn có cùng kích thước.

Leading Zeros (Số lượng bit 0 đầu tiên)

- Thuật toán đếm số **bit 0 đứng trước đầu tiên** trong các giá trị băm.
- Một tập dữ liệu có nhiều giá trị duy nhất sẽ tạo ra nhiều chuỗi hash khác nhau, trong đó có những chuỗi có nhiều số 0 đứng đầu hơn.
- Dựa vào số lượng bit 0 đứng trước, thuật toán có thể ước lượng số lượng giá trị duy nhất.

Ví dụ trong việc đếm số lượng user_id duy nhất trong 1 table. User_id ở từng dòng sẽ được mã hóa ngẫu nhiên thành các dãy bit bằng một hàm băm như MurmurHash hoặc SipHash để chuyển đổi giá trị đầu vào thành một chuỗi bit có phân bố gần như ngẫu nhiên.Xác suất xuất hiện một số lượng bit 0 liên tiếp đầu tiên trong một dãy số nhị phân là 1/(2^k), với k là số bit 0 liên tiếp đầu tiên.Nên nếu thấy giá trị hash nào có 10 bit 0 đầu tiên, hệ thống sẽ ước tính có khoảng 2^{10} (≈ 1024) user_id khác nhau.