**Từ SQL đến NoSQL: Mô hình hóa dữ liệu trong Amazon DynamoDB**

Dựa trên phân tích trong [**Part 1**](https://aws.amazon.com/blogs/database/sql-to-nosql-planning-your-application-migration-to-amazon-dynamodb/)của chuỗi bài viết, nơi chúng tôi đã xem xét cấu trúc cơ sở dữ liệu hiện tại và các mô hình truy cập, bài viết này tập trung vào việc thiết kế các mô hình dữ liệu hiệu quả cho  [Amazon DynamoDB](https://aws.amazon.com/dynamodb/).

DynamoDB cung cấp các cách tiếp cận riêng biệt trong việc mô hình hóa dữ liệu, phù hợp với yêu cầu cụ thể và mô hình sử dụng của ứng dụng.

Một mô hình dữ liệu được thiết kế tốt sẽ hỗ trợ hiệu suất tối ưu và chi phí hợp lý trong DynamoDB. Ví dụ nền tảng mạng xã hội của chúng tôi cho thấy cách các phương pháp mô hình hóa khác nhau có thể ảnh hưởng đến cả hiệu năng ứng dụng và chi phí vận hành.

Trong bài viết này, chúng tôi khám phá các chiến lược thiết kế mô hình dữ liệu DynamoDB, bao gồm xác định thực thể, các quyết định thiết kế bảng, và các cách tiếp cận trong mô hình hóa mối quan hệ. Chúng tôi cũng xem xét các tình huống thực tế để so sánh các chiến lược mô hình hóa khác nhau, giúp bạn đưa ra quyết định phù hợp với trường hợp sử dụng cụ thể của mình.

**Xác định các thực thể (Identify entities)**

Xác định các **item trong DynamoDB**, các thuộc tính (attribute) của chúng và kiểu dữ liệu tương ứng. Mặc dù các yếu tố này thường **tương ứng với schema cơ sở dữ liệu hiện tại**, bạn nên **điều chỉnh để phù hợp với mô hình hóa dữ liệu trong DynamoDB**. Hãy tập trung vào việc **tối ưu hóa cho mô hình truy cập của ứng dụng**, thay vì cố gắng sao chép cấu trúc schema hiện có một cách chính xác. Cân nhắc các yếu tố sau:

* **Thực thể cốt lõi (Core entities)** – Bắt đầu bằng cách liệt kê các thực thể chính trong nghiệp vụ, dựa trên cách ứng dụng truy cập và quản lý dữ liệu.  
  Ví dụ: Trong một ứng dụng mạng xã hội, các thực thể chính bao gồm **bài viết (post)**, **người dùng (user)** và **bình luận (comment)** — đây là các thành phần dữ liệu mà ứng dụng thường xuyên tương tác.
* **Thực thể hỗ trợ (Supporting entities)** – Xác định thêm các thực thể cần thiết để hỗ trợ chức năng của ứng dụng, chẳng hạn như:
  + Các thực thể dùng để **theo dõi số liệu và thống kê**;
  + Các thực thể để **quản lý trạng thái của ứng dụng**;
  + Các thực thể hỗ trợ **yêu cầu truy cập đặc thù**.
* **Thuộc tính (Attributes)** – Với mỗi thực thể, hãy liệt kê các thuộc tính cần thiết dựa trên nhu cầu thực tế của ứng dụng, **không phụ thuộc vào cấu trúc bảng hiện có**. Việc hiểu rõ đặc điểm của từng thuộc tính, kiểu dữ liệu và cách chúng được sử dụng trong ứng dụng sẽ giúp bạn lên kế hoạch biểu diễn chúng trong DynamoDB.  
  Ví dụ: Trường datetime có thể được ánh xạ thành **chuỗi ISO** hoặc **số epoch**, tùy theo yêu cầu truy vấn và sắp xếp.

**Quyết định thiết kế bảng (Decide on table design)**

Sau khi xác định các item trong DynamoDB, chúng tôi tiến hành đánh giá việc nên sử dụng thiết kế **single-table** (một bảng) hay **multi-table** (nhiều bảng). Thiết kế multi-table đơn giản hơn để triển khai và thường được **khuyến nghị sử dụng ở giai đoạn bắt đầu** với hầu hết ứng dụng.

Tuy nhiên, đối với **ứng dụng mạng xã hội** của chúng tôi, chúng tôi đã chọn **thiết kế một bảng duy nhất cho mỗi microservice** dựa trên phân tích nhu cầu như sau:

* **Đặc điểm ứng dụng** – Dữ liệu có tính liên kết cao, thường xuyên cần truy xuất các item liên quan cùng lúc.  
  Ví dụ: Khi hiển thị một bài viết, cần lấy đồng thời các bình luận và thông tin người dùng trong một thao tác duy nhất.  
  Ứng dụng chủ yếu xử lý dữ liệu giao dịch, **không có khối lượng lớn dữ liệu phân tích, lịch sử hay kiểm toán**. Các đặc điểm này phù hợp tự nhiên với **thiết kế single-table**.
* **Hiệu năng và chi phí** – Thiết kế single-table đảm bảo hiệu suất ổn định cho các truy vấn phức tạp liên quan đến nhiều item, vì **toàn bộ dữ liệu nằm trong cùng một bảng**.  
  Về quản lý dung lượng, chúng tôi bắt đầu với chế độ **On-Demand**, phù hợp với hầu hết workloads nhờ khả năng **tự động scale và vận hành đơn giản**.  
  Sau khi đánh giá được mô hình sử dụng và khối lượng truy cập ổn định, chúng tôi chuyển sang chế độ **Provisioned** để tối ưu chi phí.  
  Với thiết kế single-table, việc **lập kế hoạch công suất tập trung và phân bổ RCU/WCU hiệu quả** giúp tận dụng tối đa lợi ích của Provisioned mode.  
  Về lưu trữ, đặc điểm dữ liệu của chúng tôi không phát sinh bất kỳ vấn đề nào đáng kể để phải tách ra thành nhiều bảng riêng.
* **Lợi ích vận hành** – Việc quản lý một bảng duy nhất cho mỗi service đã giúp **đơn giản hóa quy trình theo dõi, lập kế hoạch dung lượng và tiến hóa mô hình dữ liệu**, từ đó **giảm tải vận hành**.

Tác động của lựa chọn thiết kế bảng sẽ khác nhau tùy theo **mô hình truy cập và đặc điểm dữ liệu** của bạn. Chúng tôi khuyến nghị nên **thử nghiệm cả hai phương pháp với workloads đại diện** để đánh giá chính xác về hiệu suất và chi phí cho trường hợp sử dụng của bạn.

Để biết thêm chi tiết, tham khảo tài liệu: [**Single-table vs. multi-table design in Amazon DynamoDB**](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-general-nosql-design.html)

**Xác định khóa phân vùng và khóa sắp xếp (Define partition and sort keys)**

Xác định các **mô hình truy cập chính** cho dữ liệu, dựa trên những hiểu biết thu được từ phân tích các truy vấn SQL. Việc này giúp bạn xác định khóa phân vùng (partition key) phù hợp cho các bảng DynamoDB.

Sử dụng thông tin từ **các mệnh đề ORDER BY và TOP** trong truy vấn SQL để lựa chọn khóa sắp xếp (sort key) — bao gồm cả việc sử dụng **khóa sắp xếp tổng hợp (composite sort key)** nếu cần sắp xếp theo nhiều cấp độ.

**Mô hình hóa mối quan hệ giữa các thực thể (Model entity relationships)**

Dù DynamoDB có **schema linh hoạt**, cho phép mô hình hóa khác với các cơ sở dữ liệu quan hệ truyền thống, bạn vẫn nên cân nhắc một số chiến lược sau để lựa chọn phương án tối ưu:

**Single item**

Cách tiếp cận "single item" có thể hiệu quả nếu một thực thể và toàn bộ dữ liệu liên quan **nằm gọn trong một item duy nhất** (tối đa 400 KB).

* Ưu điểm: Tốc độ đọc rất nhanh vì toàn bộ dữ liệu liên quan nằm cùng một chỗ.
* Nhược điểm: Mỗi thay đổi đều yêu cầu cập nhật lại toàn bộ item, dẫn đến **tăng chi phí ghi (WCU)**. Tần suất ghi càng nhiều thì chi phí và độ phức tạp trong việc giữ nhất quán dữ liệu càng cao.

Cách tiếp cận này phù hợp nhất cho các ứng dụng có **khối lượng đọc cao**, trong đó dữ liệu cha và con thường được truy xuất cùng lúc, và lợi ích từ việc tiết kiệm **chi phí đọc (RCU)** vượt trội hơn so với chi phí ghi.

📌 Lưu ý: Khi muốn **lọc dữ liệu con** bên trong một item, chỉ có thể thực hiện qua **filter expression hoặc lọc phía client**, điều này **không giúp giảm RCU** vì DynamoDB vẫn phải đọc toàn bộ item trước khi lọc.

Việc hiểu rõ các **đánh đổi** trong mô hình này là rất quan trọng để đánh giá xem đây có phải chiến lược phù hợp cho trường hợp sử dụng của bạn hay không — giữa lợi ích từ đọc nhanh và chi phí ghi cao hơn cùng với khó khăn khi lọc dữ liệu con.

**Vertical partitioning**

"Vertical partitioning" trong DynamoDB phù hợp cho các truy vấn nhắm mục tiêu cụ thể vào một số thuộc tính. Mô hình này lưu trữ các item liên quan với **cùng partition key** nhưng **khác sort key**, sắp xếp liền kề nhau.

Một số lợi ích:

* **Truy vấn linh hoạt** – Truy xuất riêng các item con hoặc kết hợp với item cha một cách hiệu quả.
* **Kiểm soát ghi chi tiết** – Cập nhật riêng từng item con mà không cần ghi lại toàn bộ item cha.

Tuy nhiên, cách tiếp cận này khiến việc **lọc dữ liệu qua nhiều thực thể** trở nên phức tạp hơn.  
→ Để lọc cả thuộc tính của cha và con, cần **de-normalize (phi chuẩn hóa)** một số thuộc tính từ thực thể cha vào các item con.

Ví dụ: Nếu bạn muốn tìm **tất cả bài viết dạng video của những người dùng đang hoạt động**, cần lọc theo:

* status = active (thuộc tính của người dùng — cha)
* media\_type = video (thuộc tính của bài viết — con)

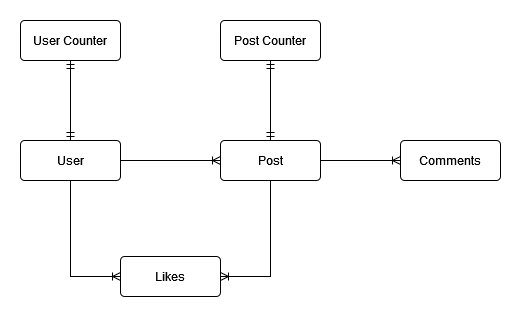
Giải pháp là thêm user\_status vào mỗi bản ghi bài viết.  
Tuy điều này **giúp truy vấn dễ dàng hơn**, nhưng nó cũng có **mặt trái**:

* Khi user\_status thay đổi, bạn cần cập nhật toàn bộ các bài viết liên quan.
* Điều này làm tăng số lượng thao tác ghi, kéo theo chi phí vận hành.

Việc đánh đổi giữa **đơn giản hóa truy vấn** và **hiệu quả ghi** là một yếu tố phổ biến trong thiết kế DynamoDB.

✅ Mấu chốt là phải **phân tích kỹ mô hình truy cập**, từ đó cân bằng giữa:

* Linh hoạt khi truy vấn,
* Dễ quản lý khi ghi dữ liệu,
* Và chi phí phát sinh từ việc phi chuẩn hóa.



**Cân nhắc về kích thước item**

Khi thiết kế mô hình dữ liệu trong DynamoDB, bạn nên ước lượng kích thước của các item. DynamoDB giới hạn kích thước tối đa **400 KB cho mỗi item**, vì vậy việc hiểu rõ kích thước trung bình và tối đa của dữ liệu là rất quan trọng.

Với mô hình **người dùng và bài viết** trong ứng dụng mạng xã hội của chúng tôi, hãy xem xét các yếu tố sau:

* Ước lượng **kích thước trung bình của dữ liệu hồ sơ người dùng**
* Tính toán **số lượng bài viết trung bình mỗi người dùng** và **kích thước điển hình của từng bài viết**

Những ước tính này sẽ giúp định hướng quá trình ra quyết định. Vì **RCU (Read Capacity Unit)** và **WCU (Write Capacity Unit)** tỷ lệ thuận với kích thước item, nên việc đánh giá kích thước trung bình của item là rất cần thiết để thiết kế một chiến lược mô hình dữ liệu tối ưu.

**Mô hình truy cập**

Hiểu rõ mô hình truy cập của ứng dụng sẽ giúp bạn thiết kế một mô hình DynamoDB hiệu quả. Hãy cân nhắc các câu hỏi sau:

* Bạn có cần **truy xuất người dùng dựa trên bộ lọc áp dụng cho bài viết của họ** không?
* Có truy vấn nào **lọc đồng thời cả dữ liệu hồ sơ người dùng và bài viết** không?
* Bạn có cần **truy cập nhanh N bài viết gần nhất của một người dùng** không?
* Bạn có cần **lấy N bài viết có nhiều bình luận nhất** không?

Các yêu cầu truy cập này sẽ định hướng việc lựa chọn **partition key**, **sort key** và **secondary index** nhằm tối ưu hóa hiệu năng.

Ngoài ra, tiêu chí lọc cũng giúp xác định **tính khả thi của phương pháp single-item**. Ví dụ, nếu cần xác định người dùng dựa trên bộ lọc của bài viết, thì **phương pháp single-item có thể không hiệu quả**, và bạn sẽ cần cân nhắc các chiến lược mô hình dữ liệu thay thế.

**Chỉ số sử dụng (Usage metrics)**

Phân tích mô hình sử dụng đọc/ghi của dữ liệu sẽ giúp xác định chiến lược tối ưu. Hãy đặt ra các câu hỏi sau để hiểu tỷ lệ đọc so với ghi:

* **Tần suất xem bài viết cùng với hồ sơ người dùng** là bao nhiêu?
* **Tần suất cập nhật bài viết so với thay đổi hồ sơ người dùng** là như thế nào?
* Những **thuộc tính nào trong hồ sơ người dùng và bài viết thường xuyên thay đổi nhất**?
* Các **bộ đếm trong bài viết** (như lượt thích, lượt chia sẻ) được đọc **bao nhiêu lần**?
* **Tần suất cập nhật bộ đếm** so với cập nhật nội dung bài viết như thế nào?

**Chọn chiến lược mô hình hóa dữ liệu**

Sau khi đã thu thập đầy đủ thông tin về kích thước item, mô hình truy cập và các chỉ số sử dụng, phần tiếp theo sẽ phân tích cách sử dụng các dữ liệu này để **đánh giá và lựa chọn chiến lược mô hình hóa dữ liệu phù hợp nhất** cho ứng dụng mạng xã hội trên DynamoDB.

**Tình huống 1: Mối quan hệ 1:N**

Bảng sau minh họa một mối quan hệ 1:N giữa **người dùng** và **bài viết**, với hai chiến lược mô hình dữ liệu được so sánh:

|  |  |  |
| --- | --- | --- |
|  | Single Item | Vertical Partition |
| Item size for 30 posts | User and posts: 20 KB + 30\*5 KB = 170 KB | User item = 20 KB First post item = 5 KB Second post item = 5 KB . . 30th post item = 5 KB |
| Number of DynamoDB API calls to get top 10 posts of a user with user information: 1,000 reads/hour | 1,000 | 2,000\* |
| RCU (eventual consistent) to read top 10 posts of a user along with user information: 1,000 reads/hour | 170 KB/4 KB = 42.5 \* 0.5 RCU = 21.25 ~ 21.5 RCU 1000 \*21.5 RCU = 21,500 RCU | User information: 20 KB ~ 2.5 RCU 10 posts: 50 KB/4 KB = 12.5 \* 0.5 RCU = 6.25 RCU ~ 6.5 RCU 1000 \* 6.5 RCU + 1000 \*2.5 RCU = 9000 RCU |
| WCU to update the user email address: 10 writes/hour | 170 WCU (updating 170 KB data) 10\*170 WCU = 1,700 WCU | 20 WCU (updating 20 KB user item only) 10\*20 WCU = 200 WCU |

**Số lượng API gọi cần thiết sẽ phụ thuộc vào cách ứng dụng được thiết kế, cụ thể là:**

* **Framework truy cập dữ liệu**
* **Thiết kế bảng DynamoDB**
* **Mô hình truy vấn**

Trong trường hợp sử dụng cụ thể này, các yếu tố đã phân tích ở trên yêu cầu **hai lệnh gọi API riêng biệt cho mỗi thao tác đọc** để lấy đủ dữ liệu cần thiết.

**Tình huống 2: Mối quan hệ 1:1**

Bảng dưới đây mô tả mối quan hệ 1:1 giữa **bài viết (mỗi bài 4 KB)** và **bộ đếm bài viết (0.5 KB)**, với hai chiến lược lưu trữ:

|  |  |  |
| --- | --- | --- |
|  | Single Item | Vertical Partition |
| Item size per post | Post + post counter: 1 KB +5 KB = 6 KB | Post item = 5 KB Post counter Item = 1 KB | |
| Number of DynamoDB API calls to read a post with post counter: 1,000 reads/hour | 1,000 | 2,000\* | |
| Read post object with post counter: 1,000 reads/hour | 6KB ~ 1 RCU 1000 \*1 RCU = 1,000 RCU | Post: 5KB ~ 1 RCU Post counter: 1 KB ~ 0.5 RCU 1000 \*0.5 +1000 \*1 = 1,500 RCU | |
| Update post counter: 10 updates/hour | 6 KB ~ 6 WCU 10 \* 5 = 60 WCU | 1 KB ~ 1 WCU 10\*1 WCU = 10 WCU | |
| Update post counter: 1,000 updates/hour | 6 KB ~ 6 WCU 1,000 \* 5 = 6,000 WCU | 1 KB ~ 1 WCU 1,000\*1 WCU = 1,000 WCU | |

***Tổng số lần gọi API cần thiết sẽ phụ thuộc vào cách ứng dụng được thiết kế*, cụ thể là framework truy cập dữ liệu, cách thiết kế bảng DynamoDB và các mẫu truy vấn.**

Trong trường hợp sử dụng cụ thể này, các yếu tố đã được phân tích ở trên yêu cầu **hai lệnh gọi API riêng biệt cho mỗi lần đọc** để truy xuất đầy đủ dữ liệu cần thiết.

**Phân tích tổng quan**

Phân tích này so sánh giữa **thiết kế single-item** và **phân mảnh dọc (vertical partitioning)** trong DynamoDB, cho cả hai loại quan hệ:

* **1:N (người dùng – bài viết)**
* **1:1 (bài viết – bộ đếm)**

Trong kịch bản 1:N, vertical partitioning yêu cầu nhiều lệnh gọi API hơn nhưng **giảm đáng kể chi phí RCU/WCU** (21.500 so với 9.000 RCU cho đọc, 1.700 so với 200 WCU cho ghi).

Tương tự, với quan hệ 1:1, dù vertical partitioning **tăng gấp đôi số lệnh gọi API**, nó mang lại **lợi ích lớn về tiết kiệm WCU** trong trường hợp cập nhật thường xuyên (6.000 so với 1.000 WCU).

Tuy nhiên, cần lưu ý rằng những kết quả này **chỉ áp dụng cho các kịch bản đã phân tích**, và **không nên áp dụng một cách chung chung** cho tất cả trường hợp.

**Kết quả cụ thể**

**1:1 (bài viết và bộ đếm):**

* **Single-item:**
  + Ít lệnh gọi API và RCU hơn khi đọc
  + Nhưng chi phí WCU cao hơn trong các lần cập nhật
* **Vertical partitioning:**
  + Cần nhiều lệnh gọi API hơn
  + Nhưng hiệu quả hơn đáng kể về WCU, đặc biệt khi cập nhật thường xuyên

**1:N (người dùng và bài viết):**

* **Single-item:**
  + Ít lệnh gọi API hơn
  + Nhưng tiêu tốn RCU đáng kể do kích thước item lớn
  + Chi phí WCU cũng rất cao khi cập nhật
* **Vertical partitioning:**
  + Số lệnh gọi API tăng gấp đôi
  + Nhưng giảm mạnh cả RCU và WCU
  + **Tiết kiệm chi phí đáng kể khi mở rộng quy mô**

**Kết luận**

Điều quan trọng không chỉ nằm ở kết quả cụ thể, mà là **quy trình phân tích** được sử dụng để đưa ra các quyết định thiết kế phù hợp.

Với bất kỳ trường hợp sử dụng nào, bạn cũng nên thực hiện một **phân tích chi tiết tương tự**, để xác định chiến lược mô hình dữ liệu tối ưu, cân bằng giữa hiệu năng và chi phí.

Một số yếu tố cần đánh giá kỹ bao gồm:

* Tần suất cập nhật
* Mô hình truy vấn
* Yêu cầu mở rộng dữ liệu
* Tính chất cụ thể của mối quan hệ giữa các thực thể

**Trong Phần 3**, chúng tôi sẽ trình bày cách điều chỉnh **lớp truy cập dữ liệu của ứng dụng** sao cho phù hợp với các mô hình dữ liệu này, giúp tận dụng tối đa các tính năng của Amazon DynamoDB.

**Thông tin tác giả**



Ramana Kiran Mannava

**Ramana** là Chuyên gia Tư vấn Cấp cao tại bộ phận Dịch vụ Chuyên nghiệp của AWS. Anh có chuyên môn trong việc hiện đại hóa các workload .NET và xây dựng các giải pháp dựa trên nền tảng đám mây AWS. Bên cạnh đó, anh cũng đam mê các công nghệ cơ sở dữ liệu và tối ưu hóa truy vấn.



Akber Rizwan Shaik

**Akber** là Chuyên gia Tư vấn Cấp cao tại bộ phận Dịch vụ Chuyên nghiệp của AWS. Anh hỗ trợ khách hàng di chuyển và hiện đại hóa các workload .NET lên AWS thông qua các giải pháp dựa trên nền tảng đám mây.



Mahesh Kumar Vemula

**Mahesh** là Chuyên gia Tư vấn Cấp cao tại bộ phận Dịch vụ Chuyên nghiệp của AWS. Anh là người đam mê kiến trúc không máy chủ (serverless) và hỗ trợ khách hàng hiện đại hóa các workload .NET của họ thông qua các giải pháp dựa trên nền tảng đám mây.