ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

-----🙞🙜🕮🙞🙜-----



**BÁO CÁO BÀI TẬP NHÓM MÔN HỌC**

**CƠ SỞ DỮ LIỆU NÂNG CAO**

**TÌM HIỂU VỀ**

**HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU MEMCACHED**

**Giảng viên hướng dẫn**: PGS.TS Nguyễn Ngọc Hoá

**Nhóm học viên** : Phạm Minh Nguyên

Nguyễn Đức Đông

Nguyễn Thị Quyền

**Nhóm** : 04

Hà Nội, tháng 12 năm 2018

1. **Đặt vấn đề**

Trong môi trường triển khai ứng dụng phân tán, một hệ thống có thể bao gồm rất nhiều các server, các node ứng dụng. Để đáp ứng được tiêu chí thời gian trả về dữ liệu cho người dùng nhanh, các ứng dụng thường làm một việc đó là lưu trữ dữ liệu đệm (cached) lên trên bộ nhớ của ứng dụng. Khi người dùng truy suất tới dữ liệu đã được lưu trữ này thì việc trả về dữ liệu cho người dùng sẽ vô cùng nhanh do không mất thời gian truy suất lại dữ liệu vào trong cơ sở dữ liệu. Tuy nhiên nếu mỗi node của ứng dụng đều lưu trữ dữ liệu đệm này thì sẽ tốn rất nhiều bộ nhớ, nên giải pháp đưa ra là sử dụng một hệ thống cơ sở dữ liệu cho phép lưu trữ dữ liệu đệm với tốc độ truy suất nhanh.

Memcached chính là một hệ thống cơ sở dữ liệu dạng Key-value với các tiêu chí như hiệu năng truy suất cao, triển khai phân tán,. Memcached giúp giải quyết bài toán lưu trữ dữ liệu tạm để tăng hiệu năng cho ứng dụng một cách hiệu quả.

Trong bài báo cáo dưới đây sẽ giải thích về cơ sở dữ liệu dạng Key-value, sau đó sẽ phân tích chi tiết về hệ thống cơ sở dữ liệu Memcached và cuối cùng là đưa ra kết luận về hệ thống cơ sở dữ liệu Memcached

1. **Tổng quan về hệ quản trị cơ sở dữ liệu Key-value**

Hệ quản trị cơ sở dữ liệu dạng Key-value là kiểu lưu trữ dữ liệu đơn giản nhất trong các loại cớ sở dữ liệu dạng NoSQL. Thông thường, các hệ quản trị cơ sở dữ liệu dạng Key-value lưu trữ dữ liệu dưới dạng key (là một chuỗi duy nhất), liên kết với value có thể ở dạng chuỗi văn bản đơn giản hoặc các tập, danh sách dữ liệu phức tạp hơn. Quá trình tìm kiếm dữ liệu thường sẽ được thực hiện thông qua key.

Các API được cung cấp cho việc truy vấn dữ liệu của các CSDL NoSQL thường cũng rất đơn giản. Về cơ bản, hầu hết các các CSDL NoSQL sẽ có các API sau:

Với kiểu lưu trữ này, ta sẽ rất dễ dàng và nhanh chóng truy xuất được thông tin của đối tượng được lưu trữ thông qua key, nhưng không hề đơn trong việc xử lý những dữ liệu phức tạp. Dễ thấy rằng ý tưởng của các hệ quản trị cơ sở dữ liệu dạng Key-value là đơn giản hóa việc lưu trữ dữ liệu, nghĩa là không cần quan tâm đến nội dung cần lưu trữ là gì. Nói cách khác, chúng lưu trữ thông tin mà không cần phải xác định lược đồ. Việc để biết được dữ liệu thực tế như thế nào sẽ được định nghĩa (mang tính tham khảo) ở phía client. Điều này làm cho phương pháp lưu trữ dữ liệu với Key-value trở nên đơn giản hơn rất nhiều trong việc xây dựng cũng như khả năng mở rộng là cực kỳ linh động và hiệu suất cho các thao tác truy vấn dữ liệu cũng cực nhanh.

Như vậy, với sự đơn giản của cách lưu trữ dạng Key – value làm cho các cơ sở dữ liệu loại này rất phù hợp với các ứng dụng cần truy xuất nhanh và khả năng mở rộng cao, chẳng hạn như là các quản lý các phiên giao dịch (session) hoặc quản lý các thông tin về giỏ hàng vì trong trường hợp này, việc biết được ID của phiên giao dịch hoặc ID của khách hàng là điều rất cần thiết. Hay việc quản lý thông tin của sản phẩm bao gồm những thông tin cơ bản, các sản phẩm liên quan, đánh giá,… sẽ được lưu trữ dưới dạng key là mã sản phẩm chẳng hạn và value là các thông tin còn lại của sản phẩm cần lưu trữ. Điều này, cho phép ta truy xuất được tất cả các thông tin về một sản phẩm chỉ thông qua mã sản phẩm cực kỳ nhanh.

1. **Giới thiệu về Memcached**

Memcached là một hệ quản trị cơ sở dữ liệu miễn phí và mã nguồn mở, có hiêu năng cao, triển khai phân tán, được dùng để tăng hiệu năng cho các ứng dụng thay thế cho việc truy vấn dữ liệu vào database. Memcached lần đầu tiên được giới thiệu năm 2003 và phiên bản mới nhất đang là bản 1.5.12 được công bố ngày 03 tháng 11 năm 2018. Memcached hiện đang được sử dụng bởi rất nhiều các ứng dụng nổi tiếng như LiveJournal, Wikipedia, Flickr, v.v.

|  |  |
| --- | --- |
| Memcached là một dạng hệ quản trị cơ sở dữ liệu dạng Key-value được lưu trên bộ nhớ cho những loại dữ liệu nhỏ tùy úy (như chuỗi, đối tượng) từ kết quả trả về của truy vấn database, kết quả của việc gọi API, v.v. Memcached được thiết kế để tăng hiệu năng ứng dụng, dễ dàng cho việc phát triển ứng dụng và giải quyết rất nhiều vấn đề trong việc lưu trữ lượng dữ liệu tạm lớn.  Đối với các ứng dụng sử dụng Memcached, các dữ liệu đệm sẽ được lưu trữ chung cho tất cả các node bên trong, vì thế nên tổng bộ nhớ lưu trữ sẽ lớn hơn so với việc sử dụng bộ nhớ đệm cho từng node. | D:\Users\dongnd3\Downloads\Blank Diagram.png  **Hình 1: Mô hình khi sử dụng Memcached** |

***Triết lý thiết kế***

Lưu trữ dữ liệu Key-Value đơn giản nghĩa là server không cần quan tâm tới dữ liệu được lưu trữ như thế nào, các phần tử được lưu theo khóa, có thời gian hết hạn, các cờ bổ sung và dữ liệu thô. Người dùng không cần biết cấu trúc dữ liệu mà chỉ cần tải lên dữ liệu đã được sắp đặt trước.

Memcached được cài đặt một phần ở client, một phần ở server. Client sẽ chọn được máy chủ nào cần để đọc và ghi dữ liệu cho một phần tử, cần phải làm gì khi không thể kết nối được tới máy chủ. Còn máy chủ sẽ biết cách để lưu trữ và lấy các phần tử, nó cũng sẽ quản lý khi nào loại bỏ và sử dụng lại bộ nhớ.

Các máy chủ độc lập với nhau, không biết về nhau. Máy chủ Memcached sẽ không trao đổi với nhau, không đồng bộ nhau, không truyền thông tin cho nhau và không tự nhân bản. Thêm các máy chủ vào sẽ tăng bộ nhớ khả dụng. Vô hiệu hóa bộ đệm được đơn giản khóa, như máy khách có thể xóa hoặc ghi đè dữ liệu trên máy chủ sở hữu dữ liệu.

Độ phức tạp thuật toán O(1), dữ liệu sẽ được xử lý gần như ngay lập tức. Đối với các máy chậm cũng có thể đáp ứng được thời gian tốt dưới 1ms. Các máy chủ cao cấp có thể xử lý hàng triệu khóa mỗi giây.

Memcached có cơ chế, được sử dụng mặc định, tự động xóa bỏ dữ liệu không được sử dụng. Những phần tử hết hạn sau một khoảng thời gian xác định sẽ bị loại khỏi bộ nhớ.

***Cách thức cài đặt***

Memcached có thể hoạt động tốt trên hầu hết các máy chủ nền tảng Linux và BSD, tuy nhiên không có bản build chính thức cho các máy chủ windows.

Để cài đặt Memcached, chúng ta có thể cài đặt từ bộ cài có sẵn hoặc cài đặt từ mã nguồn.

Đối với các hệ điều hành Debian hoặc Ubuntu:

Đối với Redhat/Fedora

Đối với cách cài đặt từ mã nguồn, trước hết bạn phải cài đặt các gói hỗ trợ

Sau đó sẽ tải về mã nguồn và thực hiện cài đặt như sau:

***Các API lệnh lưu trữ***

*set:* Dùng để lưu trữ dữ liệu, có thể được dùng để ghi đè lên dữ liệu đã tồn tại, dữ liệu được thêm vào sẽ nằm ở đầu danh sách LRU (Least Recently Used - Được sử dụng gần nhất)

*add:* Dùng để lưu trữ dữ liệu, chỉ khi nếu nó chưa tồn tại. Dữ liệu mới được thêm vào cũng sẽ nằm ở đầu danh sách LRU. Nếu dữ liệu thêm vào đã tồn tại trong bộ nhớ thì lệnh *add* sẽ báo lỗi.

*replace:* Dùng để lưu trữ dữ liệu, chỉ khi đã tồn tại trong bộ nhớ. Lệnh này hầu như không được sử dụng và chỉ tồn tại cho đầy đủ bộ giao thức

*append:* Dùng để thêm dữ liệu vào sau bản ghi đã tồn tại.

*prepend:* Giống như append nhưng dữ liệu sẽ được thêm vào trước bản ghi đã tồn tại

*cat:* Một phương thức dùng để lưu trữ dữ liệu nhưng chỉ khi dữ liệu không bị thay đổi từ lần cuối được đọc. Phương thức này rất hữu dụng trong việc xử lý việc cập nhật dữ liệu

***Các API lệnh lấy dữ liệu***

*get:* Dùng để lấy dữ liệu, có thể lấy một hoặc nhiều khóa cùng lúc

*gets:* Một phương thức thay thế lệnh *get*, để sử dụng với CAS, nó sẽ trả về một định danh CAS( một chuỗi 64bit) của phần tử. Giá trị được trả về với lệnh *cas*. Nếu CAS của phần tử đã thay đổ từ lần cuối sử dụng lệnh *gets*, nó sẽ không được lưu trữ nữa.

*delete:* Xóa phần tử khỏi bộ nhớ đệm, nếu tồn tại

*incr/decr:* Tăng hoặc giảm. Nếu một phần tử được lưu trữ dạng chuỗi 64bit số, bạn có thể chạy lệnh incr hoặc decr để thay đổi giá trị số. Bạn có thể chỉ tăng hoặc giảm bởi một số dương.

***Các API lệnh thống kê***

*stats:* Dùng để thống kê các tham số cơ bản

*stats items:* Trả về một số thông tin liên quan tới phần tử được lưu trữ trong Memcached

*stats slabs:* Trả về nhiều thông tin hơn và tập trung hơn về hiệu năng của hệ thống

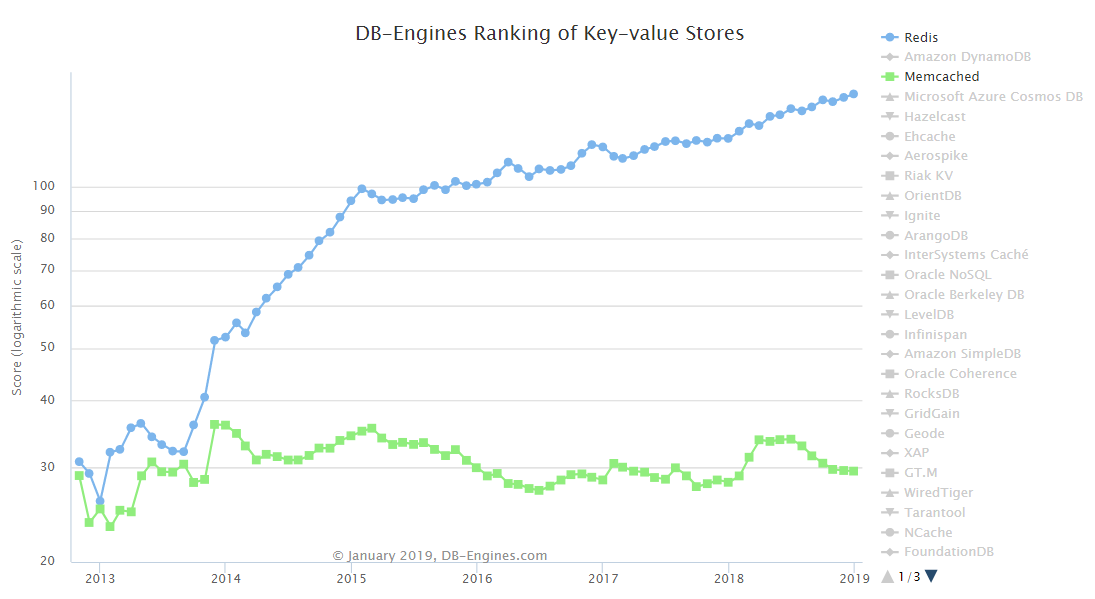
***So sánh với hệ quản trị cơ sở dữ liệu Redis***

Về tính năng thì chúng ta có thể so sánh hai hệ quản trị cơ sở dữ liệu Redis và Memcached trong bảng sau:

|  |  |  |
| --- | --- | --- |
| **Tính năng** | **Redis** | **Memcached** |
| **Lưu trong bộ nhớ** | X | X |
| **Nhân bản** | X |  |
| **Phân vùng** | X |  |
| **Nhiều kiểu dữ liệu** | X |  |
| **Xác thực** | X | X |
| **Mô hình phân phối/theo dõi** | X |  |
| **Toàn vẹn dữ liệu** | X |  |
| **Bộ nhớ ảo** | X |  |

Như chúng ta có thể thấy, Memcached tập trung chủ yếu vào việc lưu trữ dữ liệu đơn giản giống như triết lý thiết kế, còn Redis thì hỗ trợ thêm rất nhiều tính năng khác.

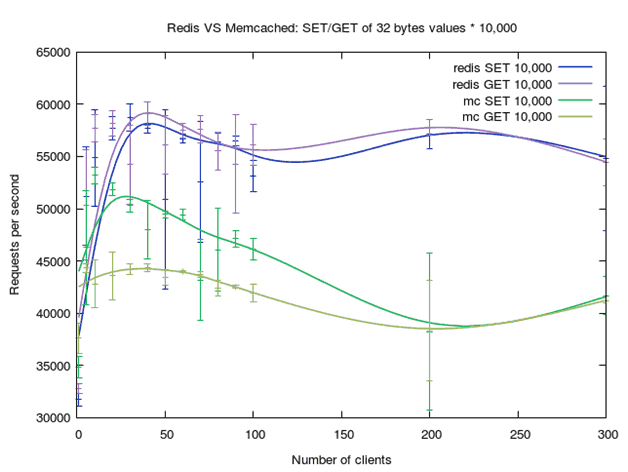
Redis có mức độ phổ biến cao hơn rất nhiều so với Memcached, hiện tại Redis đang đứng thứ 7 trong bảng xếp hạng tổng thể và thứ 1 trong bảng xếp hạng cơ sở dữ liệu Key-value của DB Engines còn Memcached đứng thứ 25 trong bảng xếp hạng tổng thể và thứ 3 trong bảng xếp hạng cơ sở dữ liệu Key-Value



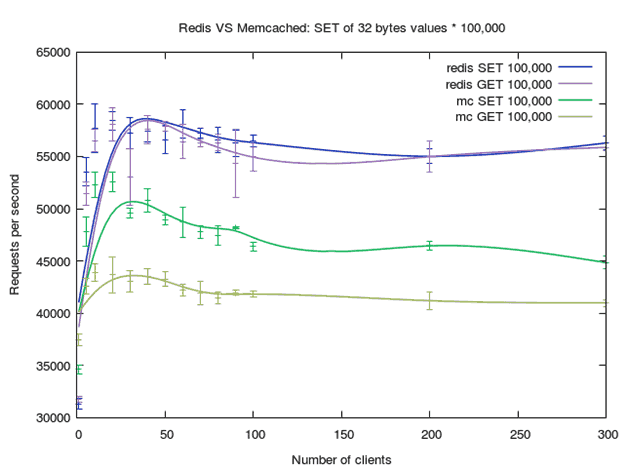
**Hình 2 - Mức độ phổ biến của Redis và Memcached**

**(Nguồn:** [**https://db-engines.com/en/ranking\_trend/key-value+store**](https://db-engines.com/en/ranking_trend/key-value+store)**)**

Thử nghiệm hiệu năng cho thấy, Redis có thể xử lý được nhiều request và nhiều client kết nối tới hơn trong cùng một môi trường



**Hình 3 - Thử nghiệm hiệu năng GET/SET x10.000 bản ghi**



**Hình 4 - Thử nghiệm hiệu năng GET/SET x100.000 bản ghi**

**(Nguồn:** [**https://www.it-cooking.com/technology/web/php/redis-vs-memcached-2018/**](https://www.it-cooking.com/technology/web/php/redis-vs-memcached-2018/)**)**

Qua thử nghiệm ta có thể thấy được Memcached và Redis đều có hiệu năng cao, tùy từng trường hợp, chúng ta có thể lựa chọn một trong hai để tích hợp. Memcached sẽ phù hợp với các ứng dụng yêu cầu sự đơn giản, hiệu năng cao, còn Redis sẽ phù hợp hơn với các ứng dụng yêu cầu sự toàn vẹn cao, hiệu năng cao và tính sẵn sàng cao

***Thực nghiệm so sánh hiệu năng Memcached và MySQL***

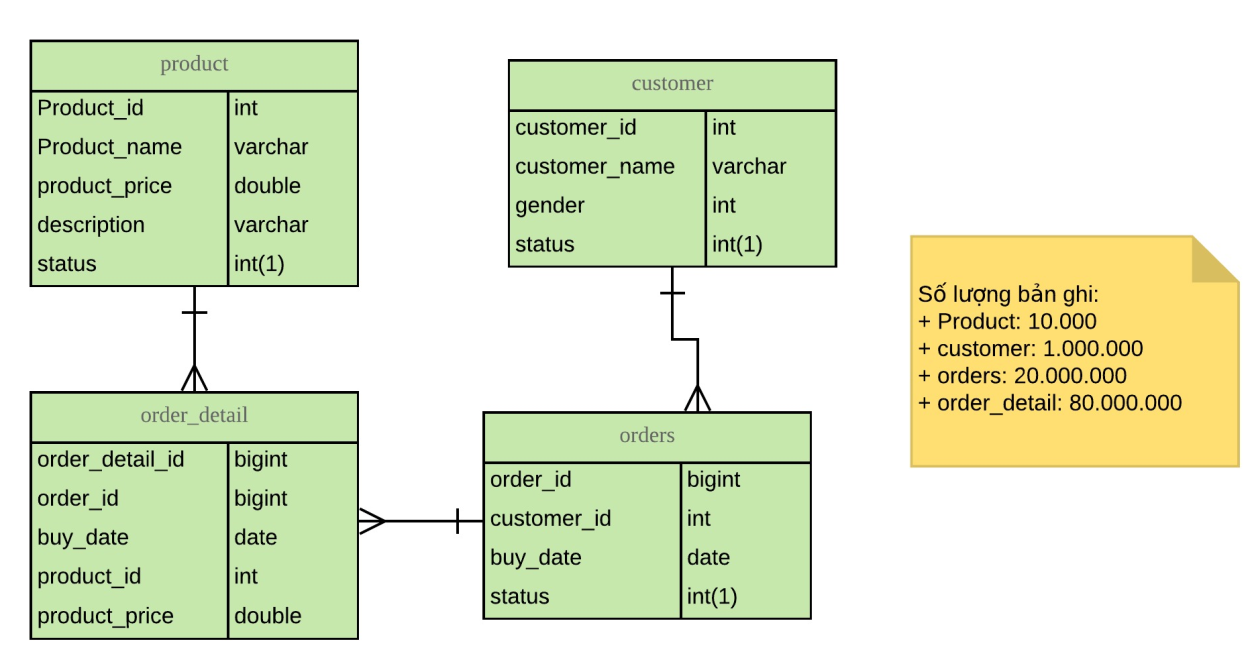
Mã nguồn của chương trình được đưa lên link sau:

**https://github.com/nddong26/csdlnc\_memcached**

Đầu vào sử dụng để so sánh:

* Memcached phiên bản 1.5.12
* MySQL phiên bản 5.7.24 (MySQL Community Server)
* Dữ liệu đầu vào ~100.000.000 bản ghi

Mô hình cấu trúc dữ liệu như hình dưới đây

******

Hình 5 - Mô hình cơ sở dữ liệu

Các bước thực hiện

* Thực hiện lấy mã nguồn từ link mã nguồn chương trình về máy
* Tạo dữ liệu test: Chạy class com.memcached.generate. GenerateDataTest sẽ sinh ra bộ dữ liệu như sau:
  + File product.txt: 10.000 bản ghi
  + File customer.txt: 1.000.000 bản ghi
  + File order.txt: 20.000.000 bản ghi
  + File order\_with\_detail.txt: 20.000.000 bản ghi (dùng để import vào memcached)
  + order\_detail.txt: 80.000.000 bản ghi
* Import dữ liệu vào MySQL:
  + Dung lượng ổ cứng yêu cầu: 20GB nếu chọn loại lưu trư thông thường, và 12GB RAM nếu chọn loại lưu trữ trên Memory
  + Tạo một cơ sở dữ liệu theo cấu trúc như hình 5
  + Thực hiện lệnh sau để import vào CSDL MySQL

#> mysql --user="{username}" --password="{password}"

mysql> use test\_db;

LOAD DATA LOCAL INFILE '/home/csdl/product.txt' INTO TABLE product;

LOAD DATA LOCAL INFILE '/home/csdl/customer.txt' INTO TABLE customer;

LOAD DATA LOCAL INFILE '/home/csdl/order.txt' INTO TABLE orders;

LOAD DATA LOCAL INFILE '/home/csdl/order\_detail.txt' INTO TABLE order\_detail;

Chú ý thay đường dẫn tới file dữ liệu test. Thời gian import vào Cơ sở dữ liệu MySQL với đầu vào ~100.000.000 bản ghi mất khoảng 10 phút với dữ liệu lưu trên ổ cứng và khoảng 8 phút 30s nếu lưu dữ liệu trên Memory

* Import dữ liệu vào Memcached
  + Dung lượng RAM yêu cầu: 50GB
  + Thực hiện chạy ứng dụng được compile từ mã nguồn của chương trình sau đó dùng trình duyệt chạy đường link sau để thực hiện import dữ liệu vào Memcached

http://{host}:{port}/{appname}/json/importMemcached

Với host, port, appname là các tham số khi triển khai ứng dụng

Cấu hình các thông tin sau trong file src/resource/config.properties

MEMCACHED\_URL: Địa chỉ memcached

MEMCACHED\_PORT: Port của memcached

MEMCACHED\_MAX\_CONNECTION: Số lượng connection tối đa

DATA\_FILE\_FOLDER: Đường dẫn tới thư mục chứa data test

Thời gian import dữ liệu vào Memcached với 100.000.000 bản ghi mất khoảng 19 phút với số lượng kết nối đồng thời dùng để import là 500

* Chạy kiểm thử hiệu năng Memcached và MySQL: Thực hiện truy vấn 10 lần liên tiếp, lấy kết quả trung bình
  + Kiểm tra hiệu năng lấy dữ liệu theo Key/ID

Thực hiện lấy dữ liệu order\_detail (80.000.000 bản ghi) từ order\_detail\_id

Bảng order\_detail trong MySQL có index trường order\_detail\_id theo khóa chính của bảng

Kiểm tra bằng cách gọi link sau trên trình duyệt, kết quả thời gian xử lý sẽ được trả về sau khi thực hiện xong

http://{host}:{port}/{appname}/json/memcached/getOrderDetailById

http://{host}:{port}/{appname}/json/mysql/getOrderDetailById

* + Kiểm tra hiệu năng lấy dữ liệu join từ nhiều bảng

Thực hiện yêu cầu tính tổng tiền của một hóa đơn. Dữ liệu tổng tiền sẽ được lấy theo mã hóa đơn (order\_id) và được tính bằng tổng số mặt hàng có trong hóa đơn.

Đối với cơ sở dữ liệu MySQL, bảng order\_detail sẽ được index thêm theo trường order\_id, bảng orders sẽ có index trường order\_id theo khóa chính của bảng. Thực hiện câu lệnh truy vấn sau để lấy tổng tiền của một hóa đơn

**select** **sum**(product\_price)

**from** test\_db.orders a, test\_db.order\_detail b

**where** a.order\_id = b.order\_id

**and** a.order\_id = ?

Đối với tính toán trên Memcached sẽ thực hiện lấy danh sách các sản phẩm của một hóa đơn theo order\_id, sau đó sẽ thực hiện lấy giá của các sản phẩm và tính tổng giá các sản phẩm

Orders customer = (Orders)

MemcachedPool.memCachedClient.get(**"ORDER\_"** + orderId);  
String orderDetailIds = customer.getOrderDetailIds();  
String[] orderDetailList = orderDetailIds.split(**"\_"**);  
Double price = 0D;  
**for** (String orderDetailId : orderDetailList) {  
 OrderDetail orderDetail = (OrderDetail)

MemcachedPool.memCachedClient.get(**"ORDERDETAIL\_"** + orderDetailId);  
 price += orderDetail.getProductPrice();  
}  
System.out.println(orderId + **":"** + price);

* Bảng kết quả tổng hợp so sánh Memcached và MySQL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tính năng/Điều kiện** | | **Memcached** | **MySQL** | **MySQL In-memory** |
| Dung lượng lưu trữ | ~100.000.000 rows | 44.2GB RAM | 12GB HDD | 9.6GB RAM |
| Thời gian import | ~100.000.000 rows | ~18 phút (500 Connection) | ~10 phút | ~8 phút 30 giây |
| Tốc độ query theo key/ID | Bảng order\_detail (80.000.000 rows) | ~1ms | ~15ms | ~1ms |
| Tốc độ query dữ liệu từ nhiều bảng | Tính tổng tiền của 1 order | ~2ms | ~38ms | ~1ms |

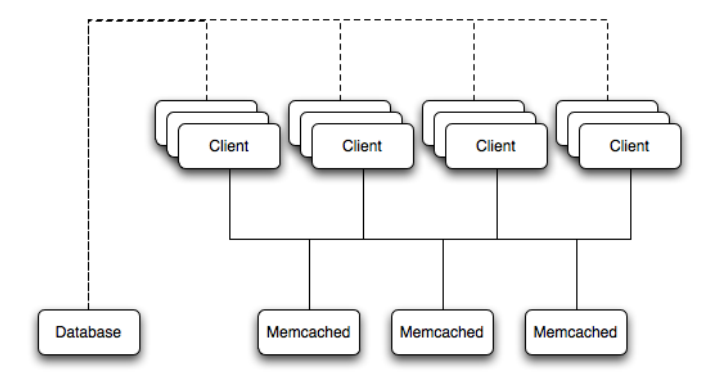
Để lưu trữ được dữ liệu khoảng 100.000.000 bản ghi thì Memcached yêu cầu rất nhiều bộ nhớ (44.2GB RAM). So sánh thời gian tốc độ lấy theo key/ID thì Memcached có hiệu năng cao hơn so với MySQL rất nhiều, gần như không có độ trễ, tương tự với việc tính tổng cũng cho hiệu năng rất cao. Chứng tỏ việc sử dụng Memcached rất hiệu quả nhưng bộ nhớ cần sử dụng cũng rất cao.

Riêng với MySQL In-memory, tốc độ truy vấn theo key/ID cũng tương đương với Memcached nhưng tốc độ tính truy vấn dữ liệu từ nhiều bảng lại nhanh hơn với Memcached mặc dù số bộ nhớ sử dụng trên Memory ít hơn so với Memcached.

***Tích hợp Memcached vào trong ứng dụng để tăng hiệu năng ứng dụng***

Giả sử chúng ta đã có một hệ thống sử dụng hệ quản trị cơ sở dữ liệu quan hệ (ví dụ MySQL) để lưu trữ dữ liệu, nhưng để năng hiệu năng đáp ứng của ứng dụng, chúng ta có thể sử dụng Memcached như trình bày dưới đây.

Mô hình ứng dụng sử dụng Memcached để tăng hiệu năng

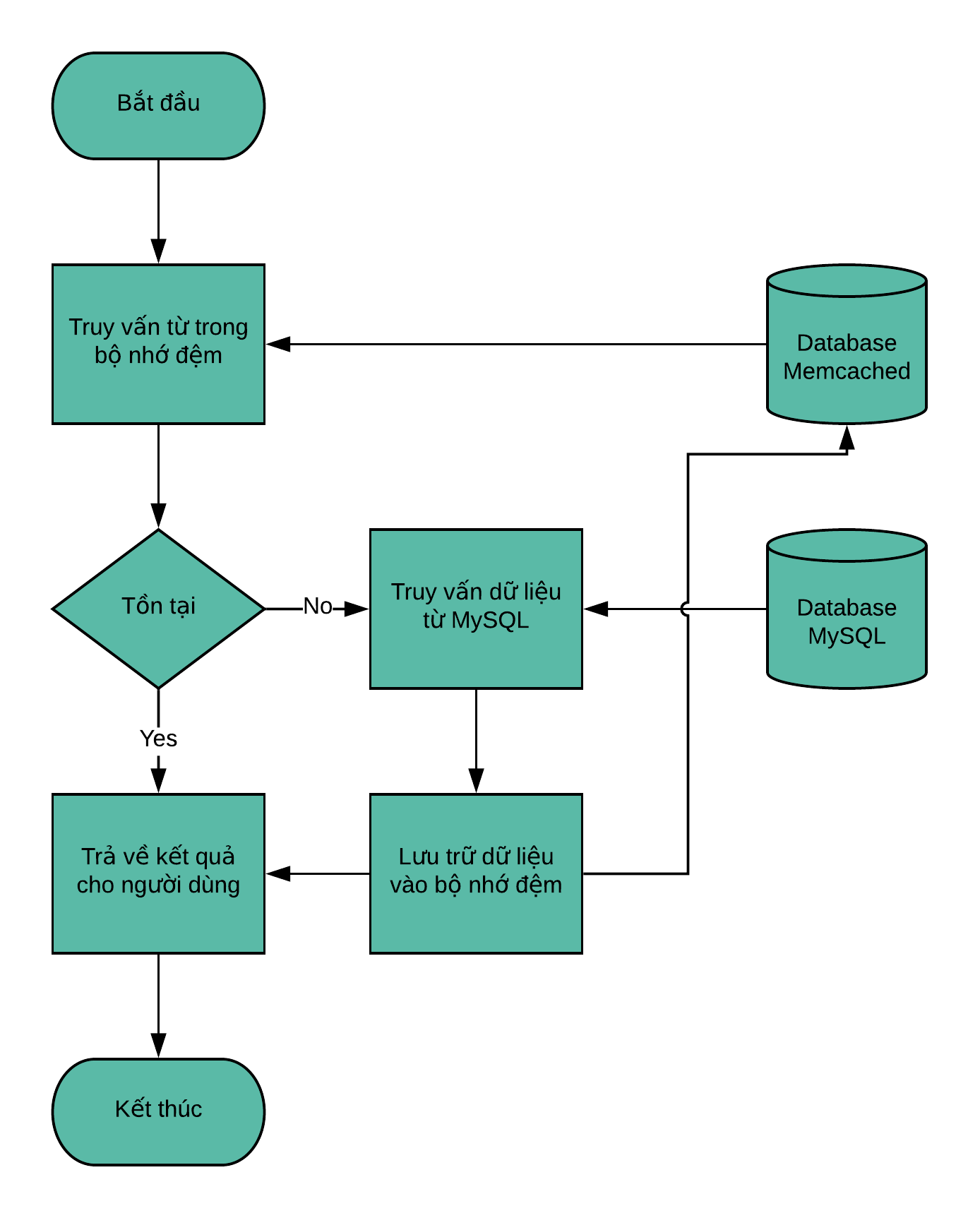


Khi sử dụng Memcached để lưu trữ dữ liệu đệm của MySQL, ứng dụng cần phải đọc dữ liệu từ trong database và tạo nên bộ khóa key-value để đưa lên bộ nhớ đệm. Sau đó các lần truy vấn dữ liệu sau này có thể được lấy trực tiếp từ trong bộ nhớ đệm.

Bởi vì MySQL cũng có việc lưu trữ bộ nhớ đệm cho các dữ liệu đã được truy vấn, như *InnoDB buffer pool* và *MySQL query cache*, nên chúng ta có thể ưu tiên lưu trữ những dữ liệu tổng hợp, như được lấy từ nhiều bảng khác nhau qua mệnh đề join, hoặc kết quả tính toán từ nhiều dòng dữ liệu.

Các bước sau sẽ mô tả việc sử dụng Memcached để lưu trữ dữ liệu đệm cho MySQL:

* Ứng dụng truy vấn dữ liệu cần lấy trong bộ nhớ đệm Memcached
* Nếu dữ liệu tồn tại 🡺 Sử dụng dữ liệu này để trả về cho người dùng
* Nếu dữ liệu không tồn tại 🡺 Truy vấn dữ liệu từ cơ sở dữ liệu MySQL và lưu trữ dữ liệu này vào bộ nhớ đệm Memcached. Và dữ liệu đã được lưu trữ vào bộ nhớ đệm này sẽ được sử dụng trong các lần truy vấn tiếp theo



**Hình 6 - Các bước sử dụng Memcached cho ứng dụng**

***Sử dụng MySQL và memcached với Java***

Có nhiều thư viện được sử dụng để hỗ trợ lập trình viên Java có thể dễ dàng thao tác với hệ quản trị cơ sở dữ liệu Memcached. Sau đây sẽ là trình bày về việc sử dụng thư viện Memcached-Java-Client (<https://github.com/gwhalin/Memcached-Java-Client/downloads>) để tích hợp hệ quản trị cơ sở dữ liệu Memcached vào ứng dụng web.

Lớp com.dâng.MemCached trong thư viện trên chính là một interface dung để giao tiếp với server Memcached. Đối tượng dung để lưu trữ trên Memcached cần được implement interface Serializable. Để sử dụng các phương thức trong lớp com.danga.MemCached, bạn cần phải sử dụng một lớp là MemCachedClient và cấu hình server Memcached thong qua SockIOPool. Thông qua đặc tả của pool, bạn có thể cài đặt địa chỉ server, số lượng kết nối hoặc tối ưu kết nối giữa ứng dụng của bạn và Memcached. Cụ thể như sau:

**protected static** MemCachedClient *mcc* = **new** MemCachedClient();  
  
**static** {  
 String[] servers  
 = {  
 **"localhost:11211"**,};  
 Integer[] weights = {1};  
 SockIOPool pool = SockIOPool.*getInstance*();  
 pool.setServers(servers);  
 pool.setWeights(weights);  
 pool.setInitConn(5);  
 pool.setMinConn(5);  
 pool.setMaxConn(500);  
 pool.setMaxIdle(1000 \* 60 \* 60 \* 6);  
 pool.initialize();  
}

Các phương thức cài đặt trong lớp com.dâng.MemCached cũng tương đương với các hàm API cung cấp bởi Memcached, cụ thể như bảng dưới đây

|  |  |
| --- | --- |
| **Phương thức lớp com.danga.MemCached** | **Phương thức trên Memcached** |
| get() | Phương thức get |
| getMulti(keys). | Trả về giá trị của nhiều khó một lúc (HashMap) |
| set() | Phương thức set |
| add() | Phương thức add |
| replace() | Phương thức replace |
| delete() | Phương thức delete |
| incr() | Phương thức incr |
| decr() | Phương thức decr |

***Một số lưu ý khi sử dụng Memcached***

Hãy chú ý tới các dữ liệu cũ đã được lưu trên Memcached và thời gian hết hạn lưu trữ của những dữ liệu đó, cần phải lựa chọn thời gian hết hạn lưu trữ của dữ liệu một cách phù hợp để tránh tình trạng tốn tài nguyên bộ nhớ không cần thiết. Việc lưu trữ một giá trị trên Memcached mãi mãi là không cần thiết, mỗi giá trị cần được cài đặt thời gian hết hạn để được giải phóng bộ nhớ tự động.

Khi sử dụng Memcached với các ứng dụng, cần phải thiết lập tường lửa để kiểm soát các kết nối của ứng dụng tới các máy chủ Memcached, và không được đưa các máy chủ Memcached ra ngoài vùng nội bộ để tránh các rủi ro bị lộ dữ liệu.

Với mỗi truy xuất dữ liệu từ Memcached, hệ thống sẽ phải mở một kết nối tới máy chủ Memcached, vì vậy việc sử dụng một hệ thống pool kết nối tới máy chủ Memcached là cần thiết để tránh việc quá nhiều kết nối cùng truy vấn tới Memcahed tại cùng một thời điểm.

Không nên sử dụng Memcached là nơi lưu trữ dữ liệu duy nhất cho ứng dụng. Dữ liệu phải luôn được lưu trữ và lưu vào Memcached từ một nguồn dữ liệu khác.

1. **Kết luận**

Để tăng hiệu năng cho ứng dụng, có rất nhiều cách khác nhau, trong đó có việc sử dụng các hệ quản trị cơ sở dữ liệu lưu trữ bộ nhớ đệm, và trong đó có hệ quản trị cơ sở dữ liệu Memcached.

Với các ưu điểm của mình như đơn giản, hiệu năng cao, dễ dàng tích hợp với các ứng dụng khác nên Memcached được sử dụng rất nhiều trong các hệ thống ứng dụng hiện nay. Tuy nhiên việc sử dụng Memcached cũng có một số hạn chế như vì dùng RAM để lưu trữ dữ liệu nên giá thành triển khai cao, mô hình triển khai của ứng dụng sẽ phức tạp thêm đôi chút và nhân viên quản trị phải vận hành thêm cụm server ứng dụng Memcached.

**Tài liệu tham khảo**

1. [**https://memcached.org/**](https://memcached.org/)
2. [**https://www.it-cooking.com/technology/web/php/redis-vs-memcached-2018/**](https://www.it-cooking.com/technology/web/php/redis-vs-memcached-2018/)
3. [**https://db-engines.com/en/ranking\_trend/key-value+store**](https://db-engines.com/en/ranking_trend/key-value+store)
4. [**https://github.com/gwhalin/Memcached-Java-Client/downloads**](https://github.com/gwhalin/Memcached-Java-Client/downloads)