­**Trường Đại học Bách Khoa Hà Nội**

**Viện Công nghệ Thông Tin và Truyền Thông**



Đồ án Tốt nghiệp Đại học

Hệ quản trị cơ sở dữ liệu theo dạng cột

Đặng Việt Hưng

Hà Nội, 12/2018

**Trường Đại học Bách Khoa Hà Nội**

**Viện Công nghệ Thông Tin và Truyền Thông**



Đồ án Tốt nghiệp Đại học

Hệ quản trị cơ sở dữ liệu theo dạng cột

|  |  |
| --- | --- |
| Sinh viên thực hiện | Đặng Việt Hưng |
| Người hướng dẫn | TS. Phạm Đăng Hải |

­­

Hà Nội, 12/2018

# Lời cam kết

Họ và tên sinh viên: Đặng Việt Hưng

Điện thoại liên lạc: 0396134566 Email: dangviethung096@gmail.com

Lớp: CNTT 2.2 K59 Hệ đào tạo: chính quy

Tôi – *Đặng Việt Hưng* – cam kết Đồ án Tốt nghiệp (ĐATN) là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *TS. Phạm Đăng Hải*. Các kết quả nêu trong ĐATN là trung thực, là thành quả của riêng tôi, không sao chép theo bất kỳ công trình nào khác. Tất cả những tham khảo trong ĐATN – bao gồm hình ảnh, bảng biểu, số liệu, và các câu từ trích dẫn – đều được ghi rõ ràng và đầy đủ nguồn gốc trong danh mục tài liệu tham khảo. Tôi xin hoàn toàn chịu trách nhiệm với dù chỉ một sao chép vi phạm quy chế của nhà trường.

|  |  |
| --- | --- |
|  | *Hà Nội, ngày tháng năm*  Tác giả ĐATN  *Họ và tên sinh viên* |

# Lời cảm ơn

Em xin gửi lời cảm ơn giáo viên hướng dẫn của em là TS. Phạm Đăng Hải đã giúp đỡ và hướng dẫn em trong quá trình làm đồ án tốt nghiệp.

Em cũng xin cảm ơn bạn bè và các thầy cô tại trường đại học Bách Khoa Hà Nội đã chỉ bảo hướng dẫn em giúp em tạo dựng các kiến thức nền tảng vững chắc trong nghề nghiệp sau này.

# Tóm tắt

Kỷ nguyên số bắt đầu khi máy tính ra đời cùng với các thiết bị lưu trữ số. Ngày nay, thay vì việc dành cả một căn phòng cho việc lưu trữ các tài liệu giấy, với một thiết bị lưu trữ như USB có thể lưu trữ toàn bộ các dữ liệu giấy. Nhưng với việc sử lưu trữ theo dạng số lại phát sinh ra vấn đề là phải làm sao để quản lý các dữ liệu đó.

Từ các vấn đề trên, hệ quản trị CSDL đã ra đời. Khi máy tính càng trở nên phổ biến và được sử dụng rộng rãi, đặc biệt trong các doanh nghiệp, tổ chức, chính phủ v.v… Việc quản lý dữ liệu tại các đơn vị này đòi hỏi rất nhiều yêu cầu như: độ bảo mật, tốc độ xử lý, và các dữ liệu được lưu trữ có thể rất đặc thù. Do đó rất nhiều HQT CSDL đã ra đời nhằm đáp ứng các yêu cầu được đặt ra. Đặc biệt với sự xuất hiện internet, các dữ liệu được chia sẻ hàng ngày là rất lớn, việc quản lý nó ngày càng khó khăn.

Với mong muốn được học hỏi các kiến thức về CSDL và có thể thiết kế nên một HQT CSDL giúp đáp ứng các yêu cầu về lưu trữ ngày nay, em đã chọn đề tài ***“Hệ quản trị cơ sở dữ liệu theo dạng cột”*** làm đề tài tốt nghiệp của mình.

# Abstract

Mục này khuyến khích sinh viên viết lại mục “Tóm tắt” đồ án tốt nghiệp ở trang trước bằng tiếng Anh. Phần này phải có đầy đủ các nội dung như trong phần tóm tắt bằng tiếng Việt.

Sinh viên không nhất thiết phải trình bày mục này. Nhưng nếu lựa chọn trình bày, sinh viên cần đảm bảo câu từ và ngữ pháp chuẩn tắc, nếu không sẽ có tác dụng ngược, gây phản cảm.

# Mục lục

[Lời cam kết iii](#_Toc533440683)

[Lời cảm ơn iv](#_Toc533440684)

[Tóm tắt v](#_Toc533440685)

[Abstract vi](#_Toc533440686)

[Mục lục vii](#_Toc533440687)

[Danh mục hình vẽ xi](#_Toc533440688)

[Danh mục bảng xii](#_Toc533440689)

[Danh mục công thức xiii](#_Toc533440690)

[Danh mục các từ viết tắt xiv](#_Toc533440691)

[Danh mục thuật ngữ xv](#_Toc533440692)

[Chương 1 Giới thiệu đề tài 1](#_Toc533440693)

[1.1 Đặt vấn đề 1](#_Toc533440694)

[1.2 Mục tiêu và phạm vi đề tài 1](#_Toc533440695)

[1.3 Định hướng giải pháp 2](#_Toc533440696)

[1.4 Bố cục đồ án 3](#_Toc533440697)

[Chương 2 NoSQL 4](#_Toc533440698)

[2.1 Khái niệm NoSQL 4](#_Toc533440699)

[2.2 Ứng dụng của NoSQL 4](#_Toc533440700)

[2.2.1 Đáp ứng số lượng người dùng lớn (Big User) 4](#_Toc533440701)

[2.2.2 The Internet of Things 5](#_Toc533440702)

[2.2.3 Big Data 5](#_Toc533440703)

[2.2.4 Cloud 6](#_Toc533440704)

[2.3 Các tính chất của NoSQL 7](#_Toc533440705)

[2.3.1 Dữ liệu luôn sẵn có 7](#_Toc533440706)

[2.3.2 Độc lập vị trí 7](#_Toc533440707)

[2.3.3 Khả năng hỗ trợ mô hình Transaction hiện đại 8](#_Toc533440708)

[2.3.4 Các mô hình dữ liệu linh hoạt 9](#_Toc533440709)

[2.3.5 Kiến trúc tốt hơn 10](#_Toc533440710)

[2.4 Ưu điểm và nhược điểm của NoSQL 11](#_Toc533440711)

[2.4.1 Ưu điểm 11](#_Toc533440712)

[2.4.2 Nhược điểm 11](#_Toc533440713)

[2.5 Mô hình nhất quán (consistency) 12](#_Toc533440714)

[2.5.1 Các loại nhất quán 13](#_Toc533440715)

[2.5.2 ACID và MVCC 13](#_Toc533440716)

[2.5.3 Two phase commit 16](#_Toc533440717)

[2.5.4 Các mức độ nhất quán 16](#_Toc533440718)

[2.5.5 Các cấp độ độc lập (Isolation) 17](#_Toc533440719)

[Chương 3 Công nghệ sử dụng 20](#_Toc533440720)

[3.1 Epoll trong linux 21](#_Toc533440721)

[3.1.1 Khái niệm epoll 21](#_Toc533440722)

[3.1.2 Các hàm trong epoll 21](#_Toc533440723)

[3.1.3 Ưu điểm và nhược điểm của epoll 21](#_Toc533440724)

[3.2 Bộ thư viện glibc 21](#_Toc533440725)

[3.2.1 Hàm open, close 21](#_Toc533440726)

[3.2.2 Hàm read, write 21](#_Toc533440727)

[3.3 Hàm băm double hashing đề xuất bởi Donald Knuth 21](#_Toc533440728)

[3.3.1 Double hashing 21](#_Toc533440729)

[Chương 4 Phát triển và triển khai ứng dụng 22](#_Toc533440730)

[4.1 Thiết kế kiến trúc 22](#_Toc533440731)

[4.1.1 Cấu trúc trong thông tin cơ sở dữ liệu 22](#_Toc533440732)

[4.1.2 Cấu trúc trong thông tin bảng 22](#_Toc533440733)

[4.1.3 Cấu trúc lưu trữ dữ liệu 22](#_Toc533440734)

[4.1.4 Lựa chọn kiến trúc phần mềm 22](#_Toc533440735)

[4.1.5 Thiết kế tổng quan 22](#_Toc533440736)

[4.1.6 Thiết kế chi tiết gói 23](#_Toc533440737)

[4.2 Thiết kế chi tiết 24](#_Toc533440738)

[4.2.1 Thiết kế giao diện 24](#_Toc533440739)

[4.2.2 Thiết kế lớp 24](#_Toc533440740)

[4.2.3 Thiết kế cơ sở dữ liệu 25](#_Toc533440741)

[4.3 Xây dựng ứng dụng 25](#_Toc533440742)

[4.3.1 Thư viện và công cụ sử dụng 25](#_Toc533440743)

[4.3.2 Kết quả đạt được 25](#_Toc533440744)

[4.3.3 Minh hoạ các chức năng chính 26](#_Toc533440745)

[4.4 Kiểm thử 26](#_Toc533440746)

[4.5 Triển khai 26](#_Toc533440747)

[Chương 5 Các giải pháp và đóng góp nổi bật 27](#_Toc533440748)

[Chương 6 Kết luận và hướng phát triển 28](#_Toc533440749)

[6.1 Kết luận 28](#_Toc533440750)

[6.2 Hướng phát triển 28](#_Toc533440751)

[Tài liệu tham khảo 29](#_Toc533440752)

[Phụ lục A-1](#_Toc533440753)

# Danh mục hình vẽ

[Hình 1. Định lý CAP 10](#_Toc533440674)

[Hình 2. Multi-version concurrency control (MVCC) 15](#_Toc533440675)

[**Hình 3** Ví dụ biểu đồ phụ thuộc gói 23](#_Toc533440676)

[**Hình 4** Ví dụ thiết kế gói 24](#_Toc533440677)

[**Hình 5** Ví dụ hình vẽ A-4](#_Toc533440678)

# Danh mục bảng

[**Bảng 1** Danh sách thư viện và công cụ sử dụng 11](#_Toc529131900)

[**Bảng 2** Ví dụ sử dụng bảng A-3](#_Toc529131901)

# Danh mục công thức

[**Công thức 1** Khai triển Newton A-5](#_Toc529131902)

Lưu ý: Nếu ĐATN có ít hơn ba công thức toán học, sinh viên có thể xóa bỏ mục này.

# Danh mục các từ viết tắt

|  |  |
| --- | --- |
| API | Application Programming Interface  Giao diện lập trình ứng dụng |
| **CSDL** | Cơ sở dữ liệu |
| **HQT CSDL** | Hệ quản trị cơ sở dữ liệu |
| **SQL** | Structured Query Language  Ngôn ngữ truy vấn cấu trúc |
| **RDBMS** | Relational database management system  Cơ sở dữ liệu quan hệ |
| **IoTs** | Internet of Things  Vạn vật kết nối |
| **SaaS** | Software as a Service  Phần mềm được coi như là một dịch vụ trong gói dành cho các đối tượng người dùng |
| **ACID** | Atomicity, Consistency, Isolation, Durability  Nguyên tố, Nhất quán, Độc lập, Bền vững |
| **MVCC** | Multi-version concurrent control |
| **IPC** | Inter-Process Communication  Kỹ thuật truyền thông giữa các tiến trình |
| **fd** | file descriptor |
| **I/O** | Input/Output  Vào/Ra trong hệ thống Linux |

# Danh mục thuật ngữ

|  |  |
| --- | --- |
| NoSQL | Cơ sở dữ liệu phi quan hệ |
| **Request** | Thao tác yêu cầu tới data nhằm thực hiện một hành động nhất định |
| **Consitency** | Sự nhất quán trong cơ sở dữ liệu |

# Giới thiệu đề tài

## Đặt vấn đề

Trước khi có sự xuất hiện của máy tính, việc lưu trữ các dữ liệu đều được thực hiện trên giấy, điều này mang lại nhiều sự khó khăn như: (i) tốn nhiều không gian để lưu trữ dữ liệu, (ii) thực hiện sao chép, tìm kiếm chỉnh sửa dữ liệu khó khăn, (iii) dễ bị mất mát dữ liệu.

Với sự xuất hiện của máy tính, việc lưu trữ dữ liệu trên máy tính đem lại rất nhiều lợi ích nổi bật như: (i) khả năng lưu trữ lớn chỉ với một không gian nhỏ, (ii) truy cập vào dữ liệu một cách nhanh chóng hiệu quả. Nhưng khi dữ liệu càng lớn thì việc quản lý, truy cập hoặc thực hiện các thao tác cơ bản như: (i) thêm, (ii) tìm kiếm, (iii) sửa, (iv) xóa càng trở nên khó khăn. Vì vậy từ những năm 60 của thế kỉ XX, khi mà máy tính được sử dụng nhiều trong các tổ chức, cơ quan thì các lập trình viên đã xây dựng các chương trình giúp quản lý dữ liệu. Và khái niệm “*cơ sở dữ liệu*” (CSDL), “*hệ quản trị cơ sở dữ liệu*” (HQT CSDL) được sử dụng nhằm ám chỉ các chương trình có khả năng quản lý dữ liệu. Các mô hình về việc xây dựng CSDL cũng được đưa ra như: (i) CSDL quan hệ, (ii) CSDL hướng đối tượng.

Với sự bùng nổ của internet và sự phổ cập của máy tính, các dữ liệu dễ dàng được số hóa và chúng ngày càng trở nên khổng lồ, đặc biệt với việc xuất hiện của các tập dữ liệu lớn (Big Data), điều này đặt ra các bài toàn cho việc khai thác và biến các dữ liệu này thành các tri thức giúp ích cho con người. Các “*hệ quản trị cơ sở dữ liệu*” được xây dựng theo các mô hình cũ không còn thích hợp nữa khi gây ra sự khó khăn như trong việc tìm kiếm hoặc phân tán dữ liệu. Vì vậy, nhằm đáp ứng các yêu cầu trên nhiều mô hình CSDL được ra đời. Trong đó phải kể đến các “*cơ sở dữ liệu phi quan hệ (NoSQL)*”, đây được coi là CSDL thế hệ mới nhằm đáp ứng cho các bài toán về Big Data, về lưu trữ phân tán v.v…

## Mục tiêu và phạm vi đề tài

Hiện nay, đã có một số “*cơ sở dữ liệu phi quan hệ*” được phát triển dựa và sử dụng rộng rãi như: (i) mongoDB, (ii) riak, (iii) Apache Cassandra, (iv) Amazon DynamoDB. Tuy nhiên hiện tại mỗi CSDL này sử dụng các cấu trúc lưu trữ là khác nhau và chúng cũng phù hợp đối với từng yêu cầu nhất định như: (i) hiệu năng cao, (ii) khả năng mở rộng, (iii) cấu trúc linh hoạt.

Tuy vậy CSDL NoSQL vẫn còn rất nhiều hạn chế như: (i) Không có được sự hỗ trợ tốt nhất do đa phần là mã nguồn mở, (ii) chưa thực sự được tin cậy đối với đa phần doanh nghiệp, (iii) hạn chế về tri thức nghiệp vụ của nhân viên kỹ thuật do đây là các công nghệ mới và thay đổi từng ngày, (iv) vấn đề với sự tương thích do chưa có được một tiêu chuẩn nhất quán.

Với mong muốn được nghiên cứu, tìm hiểu và xây dựng một trong các mô hình của CSDL NoSQL, em đã lựa chọn xây dựng một *cơ sở dữ liệu được theo dạng cột*, là một trong những mô hình sử dụng cấu trúc dữ liệu lưu trữ khác so với *cơ sở dữ liệu quan hệ* truyền thống.

Đề tài này sẽ tập trung vào việc thiết kế mô hình lưu trữ dữ liệu theo dạng cột, cung cấp các thao tác cơ bản đối với CSDL là (i) tìm kiếm, (ii) thêm, (iii) sửa, (iv) xóa. Và sử dụng cấu trúc server, client cho CSDL nhằm cung cấp khả năng đáp ứng đối với nhiều người dùng và từ đó có thể xây dựng lên CSDL phân tán.

## Định hướng giải pháp

Trong các mô hình áp dụng vào CSDL NoSQL, *cơ sở dữ liệu dạng cột* sẽ giúp tăng tốc độ truy cập vào dữ liệu tại một trường nhất định giúp giảm chi phí cho việc truy cập vào các trường giá trị dư thừa trong CSDL.

Đồng thời, đề tài này sẽ sử dụng bảng băm để lưu trữ các trường giá trị nhằm tăng giảm thiểu tối đa việc tìm kiếm và thêm. Hàm băm được sử dụng là hàm băm doube hashing sử dụng open address để giải quyết vấn đề đụng độ được đề xuất bởi Donald Knuth. Chi tiết về kỹ thuật băm sẽ được giải thích chi tiết trong chương 5.

Với mô hình client-server cho database sẽ sử dụng API của Linux là epoll cho phía server nhằm đáp ứng được số lượng client nhiều và tăng tốc độ xử lý đối với các request tới từ các client.

Từ phương pháp trên, mô hình CSDL dạng cột sẽ được thiết kế và chạy thử nghiệm. Chi tiết các trường hợp thử nghiệm sẽ được trình bày tại chương 6.

## Bố cục đồ án

Phần còn lại của báo cáo đồ án tốt nghiệp này được tổ chức như sau.

Chương 2 trình bày về v.v.

Trong Chương 3, em/tôi giới thiệu về v.v.

# NoSQL

## Khái niệm NoSQL

NoSQL là một thuật ngữ dùng làm tên gọi chung cho các HQT CSDL cung cấp các cơ chế lưu trữ và truy xuất dữ liệu sử dụng mô hình nhất quán lỏng lẻo hơn các HQT CSDL quan hệ truyền thống. Điều này có nghĩa là dữ liệu của các CSDL thuộc NoSQL sẽ không được xây dựng trên các bảng (table), không có các quan hệ khóa chính (primary key) – khóa ngoại (foreign key), và không sử dụng SQL làm ngôn ngữ truy vấn dữ liệu. NoSQL là CSDL phân tán được thiết kế dành cho các hệ thống cần khả năng mở rộng lưu trữ lớn, tính sẵn sàng (availability) và hiệu năng (performance) cao.

Từ NoSQL thường được hiểu và chấp nhận rộng rãi là Not only SQL (không chỉ có SQL), điều này nhằm nhấn mạnh rằng việc nó cũng hỗ trợ ngôn ngữ SQL.

Sự ra đời của các CSDL NoSQL đã tạo nên một cuộc cách mạng về công nghệ lưu trữ trong thời đại của Web 2.0, nơi mà nhu cầu về tính sẵn sàng và hiệu năng cao rất được xem trọng. Bên cạnh đó, các mạng dịch vụ cộng đồng phát triển mạnh mẽ cho phép người dùng tạo ra hàng tỷ nội dung trên web. Do đó, dữ liệu lớn rất nhanh vượt qua cả những giới hạn về phần cứng và cần giải quyết bằng bài toán phân tán.

## Ứng dụng của NoSQL

### Đáp ứng số lượng người dùng lớn (Big User)

Khi internet mới bắt đầu phổ biến, việc một ứng dụng có khoảng 1000 người dùng thì rất phổ biến, nhưng nếu nâng con số này lên 10.000 hay thậm chí 100.000 thì đó là chuyện không tưởng. Ngày nay, hầu hết các ứng dụng mới đều được lưu trữ trên cloud và có luôn có sẵn trên Internet, nơi mà các ứng dụng này phải phục vụ cho người dùng trên toàn thế giới mọi lúc mọi nơi. Có khoảng hơn 2 tỷ người được kết nối với Internet và thời gian mà họ bỏ ra cho việc online mỗi ngày cũng tăng theo thời gian. Điều này tạo ra sự bùng nổ về số lượng người dùng đồng thời. Như vậy, sẽ không có gì lạ khi một ứng dụng có tới hàng triệu người khác nhau sử dụng mỗi ngày. Số lượng người dùng lớn kết hợp với thói quen sử dụng phong phú tạo nên nhiều mô hình dữ liệu khác nhau. Điều này dẫn đến sự cần thiết của các công nghệ CSDL có khả năng mở rộng dễ dàng. Với RDBMS, nhiều nhà phát triển ứng dụng đã gặp rất nhiều khó khăn hoặc thậm chí là không thể trong việc tạo ra các ứng dụng có khả năng mở rộng linh động trong khi vẫn duy trì được hiệu năng tốt như những gì người dùng mong đợi. Và họ đã tìm đến với các giải pháp NoSQL.

### The Internet of Things

The Internet of Things có thể hiểu đơn giản là một mạng lưới gồm các đối tượng có khả năng kết nối Internet và tác động qua lại giữa các dịch vụ web. Thực tế cho thấy, số lượng các thiết bị có thể kết nối internet đồng thời có khả năng tạo ra dữ liệu đang ngày một gia tăng và trở nên phổ biến. Những thiết bị đang có mặt ở khắp mọi nơi và khá gần gũi với chúng ta, chúng có thể đang ở trong các nhà máy, công ty, bệnh viện, trường học, hay gia đình chúng ta, v.v… Chúng là điện thoại di động, máy tính bảng, các loại máy móc chuyên dụng, và còn rất nhiều nữa. Theo thời gian, những thiết bị này ngày càng được nâng cấp thành những thiết bị tiện dụng và thông minh hơn, một điển hình quen thuộc đó là những chiếc smart phone, tablet, v.v…

Tuy nhiên, dữ liệu từ xa thường thì nhỏ, bán cấu trúc (semi-structured), hoặc không có cấu trúc (unstructured) và được cập nhật liên tục. Điều này mang đến nhiều thách thức cho các RDBMS vốn đòi hỏi một lược đồ cố định (fixed shema) và dữ liệu có cấu trúc (structured data). Để giải quyết thách thức này, các doanh nghiệp có xu hướng đổi mới đang tin tưởng vào công nghệ NoSQL để mở rộng việc truy cập đồng thời cho hàng triệu thiết bị kết nối, lưu trữ hàng tỷ điểm dữ liệu đồng thời đáp ứng các yêu cầu về hiệu năng của cơ sở hạ tầng (infrastructure) và các hoạt động (operation) then chốt (mission-critical).

### Big Data

Dữ liệu đang ngày một trở nên dễ dàng nắm bắt và truy cập thông qua bên thứ ba như Facebook, Google,. . . Thông tin cá nhân của người dùng, dữ liệu định vị địa lý, những nội dung do người dùng tạo là một vài ví dụ về các mảng dữ liệu không ngừng được mở rộng đã tạo nên một nguồn dữ liệu vô cùng phong phú. Không có gì phải ngạc nhiên khi các nhà phát triển tận dụng những nguồn tài nguyên sẵn có này để làm phong phú thêm những ứng dụng hiện có hay tạo ra những cái mới từ những nguồn tài nguyên đó. Việc sử dụng các nguồn dữ liệu đang làm thay đổi nhanh chóng bản chất của truyền thông, hành vi mua sắm, quảng cáo, giải trí và việc quản lý các mối quan hệ xã hội. Như vậy, một nhà phát nếu biết tận dụng nguồn dữ liệu sẵn có này một cách hợp lý vào việc phát triển ứng dụng sẽ dễ dàng thành công, ngược lại, việc nhanh chóng đi vào thất bại là điều dễ hiểu do không nắm bắt được thị hiếu của người dùng.

Việc lưu trữ và sử dụng các loại dữ liệu khác nhau tạo ra nhu cầu về loại CSDL cũng rất riêng. Tuy nhiên, các nhà phát triển muốn có một CSDL thật linh hoạt dễ dàng chứa bất kỳ kiểu dữ liệu mới nào mà họ muốn sử dụng và không bị gián đoạn bởi những thay đổi cấu trúc của nội dung dữ liệu từ nhà cung cấp dữ liệu thứ ba. Rất nhiều những dữ liệu mới là các dữ liệu không cấu trúc hoặc bán cấu trúc, do đó, các nhà phát triển cũng cần có một CSDL có khả năng lưu trữ hiệu quả. Tiếc là, việc định nghĩa một cách cứng nhắc, phương pháp tiếp cận dựa trên lược đồ (schema) được sử dụng trong các RDBMS làm cho nó không thể nhanh chóng kết hợp với các loại dữ liệu mới được, và kết quả là không phù hợp với dữ liệu không cấu trúc và bán cấu trúc.

Như vậy, với sự tăng lên nhanh chóng của dữ liệu thì việc xử lý dữ liệu lớn đóng vai trò hết sức quan trọng đặc biệt là với các loại dữ liệu không cấu trúc và bán cấu trúc. Điều này đã đặt ra nhiều thách thức cho các RDBMS truyền thống, và dễ dàng thấy rằng, với các ràng buộc về cấu trúc dữ liệu, sử dụng các lược đồ cố định thì RBDMS đã trở nên vô cùng khó khăn để xử lý lượng dữ liệu cực lớn mà đa phần là không có cấu trúc hoặc bán cấu trúc đang gia tăng hàng ngày, hàng giờ. Trong khi đó, NoSQL cung cấp mô hình dữ liệu tốt hơn làm cho đơn giản hóa việc giao tiếp giữa CSDL và ứng dụng.

### Cloud

Trước đây, hầu hết các ứng dụng dành cho người dùng và các ứng dụng trong kinh doanh đều là các ứng dụng đơn người dùng và được cài trên một máy tính cục bộ. Các ứng dụng đa người dùng phần lớn sử dụng kiến trúc 2 tấng (2-tier) hay client-server chạy bên trong các bức tường lửa và hỗ trợ giới hạn số lượng người sử dụng.

Ngày nay, hầu hết các ứng dụng mới sử dụng kiến trúc 3 tầng, được lưu trữ và thực thi trên những đám mây, hỗ trợ một số lượng lớn người sử dụng đồng thời. Cùng với sự thay đổi này trong kiến trúc phần mềm, mô hình kinh doanh mới như là SaaS và các mô hình dựa trên quảng cáo (Advertising-based) cũng đã trở nên phổ biến hơn. Trong kiến trúc 3 lớp, ứng dụng được truy cập thông qua một trình duyệt web hoặc một ứng dụng di động được kết nối tới Internet. Trong đám mây, một cân bằng tải (load balancing) sẽ điều khiển lưu lượng truy cập đến một tầng có khả năng mở rộng theo chiều ngang (scale-out) của máy chủ web/ứng dụng mà xử lý logic của ứng dụng. Kiến trúc mở rộng theo chiều ngang tại tầng ứng dụng làm việc rất tốt. Với mỗi 10.000 (hay tùy ý) người dùng đồng thời, ta chỉ cần thêm một server khác đến tầng ứng dụng để chịu tải. Tại tầng CSDL, CSDL quan hệ từng được lựa chọn nhiều nhất, và việc sử dụng chúng ngày càng gặp nhiều vấn đề rắc rối. Bởi vì, dữ liệu của các CSDL quan hệ thường được tập trung và có xu hướng mở rộng theo chiều dọc (scale up) hơn là mở rộng theo chiều ngang (scale out). Điều này dẫn đến các CSDL rất khó phù hợp với các ứng dụng yêu cầu khả năng mở rộng động (dynamic scalability) dễ dàng. Các công nghệ NoSQL đã xây dựng với mục đích phân tán ngay từ lúc ban đầu (ground up) cộng với các kỹ thuật mở rộng theo chiều ngang, vì vậy tương thích tốt với bản chất phân tán cao của kiến trúc 3 tầng của Internet.

## Các tính chất của NoSQL

### Dữ liệu luôn sẵn có

Môi trường CSDL NoSQL thường được xây dựng với kiến trúc phân tán và nó được dự phòng cho cả chức năng và dữ liệu. Nếu một hoặc nhiều nodes bị sập, có các node khác trong hệ thống có thể tiếp tục thao tác mà không để mất dữ liệu. Điều này cho thấy khả năng chịu lỗi của hệ thống. Bằng cách này, môi trường CSDL NoSQL có khả năng duy trì tính sẵn có của dữ liệu tại một vị trí duy nhất, thông qua các trung tâm dữ liệu và trong cloud. Khi triển khai thích hợp, các hệ CSDL NoSQL có thể cung cấp khả năng mở rộng lớn cùng với hiệu năng cao mà không bao giờ bị sập. Điều này đặc biệt có lợi trong bất kì hoạt động nâng cấp hoặc cập nhật hệ thống mà cần không phải có một CSDL ngoại tuyến.

### Độc lập vị trí

"Độc lập vị trí" có nghĩa là khả năng đọc và ghi vào CSDL bất kể thao tác Input/Output (I/O) được xảy ra ở đâu và có bao nhiêu bản sao chép từ vị trí đó, như vậy, nó có thể sẵn có đối với những người dùng tại các vị trí khác. Chức năng này rất khó để thiết kế cho CSDL quan hệ. Một vài kĩ thuật có thể được sử dụng như kiến trúc master/salve hay database sharding (phân mảnh CSDL) đôi lúc có thể có được sự độc lập vị trí trong thao tác đọc nhưng không thể thao tác ghi lại là vấn đề khác, đặc biệt khi kích thước dữ liệu lớn. Một trong những trường hợp thể hiện tính ưu việt của đọc lập vị trí là khi có nhiều dịch vụ, bao gồm cả dịch vụ khách hàng tại nhiều khu vực địa lý khác nhau và cần lưu trữ cục bộ những thông tin này tại các khu vực đó nhằm tăng tốc độ truy cập.

### Khả năng hỗ trợ mô hình Transaction hiện đại

Lý thuyết về transactions xuất hiện để được thay đổi trong thời đại Internet, và nó được chứng minh răng ACID transactions không còn được xem là một yêu cầu bắt buộc trong các hệ thống CSDL. Nhận định này có vẻ cực đoan khi mà toàn vẹn giao dịch là một trong những đặc trưng của mọi hệ dữ liệu, đặc biệt là những hệ thống yêu cầu thông tin trung thực và chính xác. Tuy nhiên, điều này nói đến không phải là gây nguy hiểm cho dữ liệu, mà là một cách thức mới mà các ứng dụng hiện đại đảm bảo tính nhất quán trên toàn hệ thống phân tán.

Tính nhất quán trong HQT CSDL quan hệ được thực thi thông qua các ràng buộc khóa ngoại. Để đảm bảo tính toàn vẹn (integrity) của dữ liệu, hầu hết các HQT CSDL cổ điển đều dựa trên các giao dịch (transaction). Điều này đảm bảo tính nhất quán (consistency) trong tất cả các trường hợp quản lý dữ liệu. Những đặc tính về giao dịch này được biết đến chính là ACID (Atomicity, Consistency, Isolation, Durability).

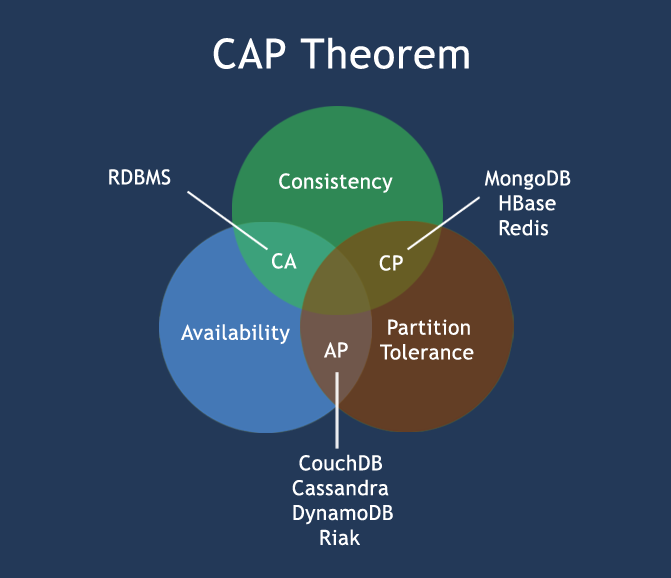
Việc mở rộng theo chiều ngang (scale out) của các hệ thống tuân thủ theo ACID đã thể hiện một số vấn đề. Những xung đột đang phát sinh giữa các khía cạnh khác nhau của tính sẵn sàng cao trong các hệ thống phân tán mà không thể giải quyết hoàn toàn được, điều này được biết với định lý CAP. Định lý CAP nói rằng trong một hệ CSDL phân tán, bạn chỉ có thể đạt được nhiều nhất hai trong ba tính chất: Consistency, Availability, Partition tolerance

Trong đó:

* *Strong Consistency* (tính nhất quán mạnh mẽ): nghĩa là cách mà hệ thống ở trong một trạng thái thống nhất sau khi thực hiện một thao tác. Một hệ thống phân tán thường được coi là nhất quán nếu sau khi cập nhật hoạt động của một vài thao tác ghi thì tất cả các thao tác đọc dữ liệu sẽ thấy được dữ liệu đã được cập nhật đó.
* *High Availability* (tính sẵn sàng cao): nghĩa là một hệ thống được thiết kế và hiện thực sao cho khi có vấn đề xảy ra với một node trong một cụm (cluster) chẳng hạn như bị hư hỏng hay được tạm ngưng để nâng cấp thì các theo tác đọc ghi dữ liệu vẫn cho phép tiếp tục thực hiện.
* *Partition Tolerance*: nghĩa là hệ thống của một cụm bất kỳ vẫn hoạt động bình thường ngay cả khi đường truyền giữa hai node bị ngắt và không liên lạc được.

### Các mô hình dữ liệu linh hoạt

Một trong những nguyên nhân chính để cuyển sang hệ QTCSDL NoSQL là do những mô hình dữ liệu linh hoạt hơn đến từ các hệ QTCSDL NoSQL. Mô hình CSDL quan hệ dựa trên mối liên hệ giữa các bảng, trong đó các bảng được xác định bởi cấu trúc cột. Tất cả chúng được tổ chức chặt chẽ và đồng nhất trong một database schema. Vấn đề bắt đầu nảy sinh với mô hình quan hệ xung quanh khả năng mở rộng và hiệu năng khi cố gắng quản lý một lượng lớn dữ liệu. Mô hình NoSQL - thường được gọi là schema-less - có thể hỗ trợ rất nhiều những trường hợp sử dụng này. Một CSDL NoSQL có thể chấp nhận mọi loại dữ liệu - có cấu trúc, bán cấu trúc và không cấu trúc- một cách dễ dàng hơn một CSDL quan hệ, khi mà CSDL quan hệ luôn dựa vào schema được xác định sẵn. Đặc điểm này của CSDL quan hệ có thể trở thành ựự cản trở cho tính linh hoạt bởi cấu trúc được định nghĩa trước một cách cứng nhắc xác định cấu trúc và cách thức tổ chức dữ liệu của RDBMS. Trong rất nhiều các ứng dụng ngày nay có khả năng thực thi các quy định về việc sử dụng dữ liệu do chính dữ liệu đó xác định, tạo một nền tảng schema-less DB là một cách khả thi.



Hình 1. Định lý CAP

Cuối cùng là vấn đề về hiệu năng đối với mô hình RDBMS, đặc biệt là khi các "wide rows" có liên quan và thực hiện nhiều thao tác update có thể làm giảm hiệu năng hệ thống một cách nghiêm trọng. Tuy nhiên, mô hình dữ liệu NoSQL có thể dễ dàng xử lý các tình huống này và mang lại hiệu năng rất cao cho cả các thao tác đọc và ghi.

### Kiến trúc tốt hơn

Một trong những lý do để dùng CSDL NoSQL là khi cần một kiến trúc thích hợp hơn cho các ứng dụng tương ứng. Một vài, nhưng không phải tất cả, các giải pháp NoSQL có thể cung cấp các kiến trúc hiện đại có thể giải quyết các loại ứng dụng đòi hỏi cao khả năng mở rộng, phân tán và tính sẵn sàng liên tục. Hỗ trợ trung tâm dữ liệu, và thông dụng hơn là hỗ trợ nhiều trung tâm dữ liệu là một trong những trường hợp bắt buộc phải sử dụng môi trường NoSQL.

## Ưu điểm và nhược điểm của NoSQL

### Ưu điểm

**Open source**: các sản phẩm nguồn mở đưa ra cho những người phát triển với nhiều lợi ích to lớn, đặc biết là việc sử dụng miễn phí. Những lợi ích khác là phần mềm nguồn mở có xu hướng sẽ là tin cậy hơn, an ninh hơn và nhanh hơn để triển khai so với các lựa chọn thay thế sở hữu độc quyền.

**Khả năng mở rộng linh hoạt**: NoSQL thay thế câu thần chú cũ của các nhà quản trị CSDL về "mở rộng phạm vi" với một thứ mới "mở rộng ra ngoài". Thay vì bổ sung thêm các máy chủ lớn hơn để điều khiển nhiều tải dữ liệu hơn, thì CSDL NoSQL cho phép một công ty phân tán tải qua nhiều máy chủ khi mà tải gia tăng.

**Các CSDL NoSQL khác nhau cho những dự án khác nhau**: MongoDB và Redis là những lựa chọn tốt cho việc lưu trữ các dữ liệu thống kê ít được đọc mà lại được viết thường xuyên, như một số đếm truy cập web chẳng hạn. Hadoop, một CSDL dạng tự do, phân tán làm tốt công việc lưu trữ các dữ liệu lớn như các con số thống kê thời tiết hoặc công việc phân tích nghiệp vụ.

**NoSQL được các hãng lớn sử dụng**: các công ty như Amazon, BBC, Facebook và Google dựa vào các CSDL NoSQL.

**NoSQL phù hợp với công nghệ đám mây**: NoSQL và đám mây là một sự trùng khớp tự nhiên. Các máy chủ ngày nay là không đắt và có thể dễ dàng mở rộng phạm vi được theo yêu cầu có sử dụng một dịch vụ như là Amazon EC2. Giống như tất cả công nghệ đám mây, EC2 dựa vào ảo hóa. Liên kết yếu của ảo hóa là sự thực thi của I/O, với bộ nhớ và CPU

các các kết nối mạnh. Các CSDL NoSQL hầu hết sử dụng bộ nhớ qua đĩa như là vị trí ghi đầu tiên. Vì thế ngăn ngừa được sự thực thi không ổn định của I/O. Và vì NoSQL lưu trữ dữ liệu thường thúc đẩy được tính mở rộng phạm vi theo chiều ngang thông qua việc ngăn chia, chúng có khả năng tận dụng được việc cung cấp mềm dẻo của đám mây.

### Nhược điểm

**Hỗ trợ không đồng đều cho các doanh nghiệp**: trong khi các nhà cung cấp chủ chốt của các RMDBMS như SQL Server, Oracle, IBM,. . . thường đưa ra sự hỗ trợ tốt cho khách hàng thì các nhà cung cấp nguồn mở mới thành lập không thể được mong đợi sẽ cung cấp hỗ tốt hơn.

**Chưa đủ "chín" cho các doanh nghiệp**: dù chúng đã được triển khai tại một số công ty lớn thì các CSDL NoSQL vẫn đối mặt với một vấn đề về sự tin cậy chính với nhiều doanh nghiệp. Vấn đề lớn của NoSQL là thiếu về độ chín muồi và các vấn đề về tính không ổn định, trong khi đó tính chín muồi, hỗ trợ đầy đủ chức năng và tính ổn định của các RDBMS được thiết lập đã từ lâu.

**Những hạn chế về tri thức nghiệp vụ**: các CSDL NoSQL không có nhiều sự đeo bám tới các công cụ BI thường được sử dụng, trong khi những yêu cầu và phân tích hiện đại đơn giản nhất thì cũng liên quan khá nhiều tới sự tinh thông về lập trình.

**Thiếu sự tinh thông**: tính mới mẻ của NoSQL có nghĩa là không có nhiều lập trình viên và người quản trị biết công nghệ này. Như vậy sẽ rất khó khăn cho các công ty tìm người với sự tinh thông phù hợp.

**Những vấn đề về tính tương thích**: không giống như các CSDL quan hệ, các CSDL NoSQL chia sẻ ít theo cách thức của các tiêu chuẩn. Mỗi CSDL NoSQL có các giao diện lập trình ứng dụng API riêng của mình, các giao diện truy vấn riêng,. . . Sự thiếu hụt các tiêu chuẩn sẽ gây ra rất nhiều khó khăn khi chuyển từ một nhà cung cấp này sang một nhà cung cấp khác nếu có nhu cầu.

## Mô hình nhất quán (consistency)

Một trong những yếu tố tạo nên sức mạnh cho NoSQL DB là sự thoát khỏi sự nhất quán ACID chặt chẽ. Mọi người thường có suy nghĩ rằng các loại CSDL NoSQL chỉ cung cấp weak consistency hoặc nhiều nhất là eventual consistency. Đây là một trong những hiểu lầm cơ bản với những hệ CSDL NoSQL. Các hệ thống CSDL NoSQL cung cấp một phạm vi consistency, bao gồm cả strict consistency.

Các mô hình nhất quán ảnh hưởng lớn tới sự song song hóa của CSDL - khả năng truy cập dữ liệu đồng thời từ người dùng - và tính sẵn sàng. Hiểu được cách hệ CSDL xử lý với consistency là cần thiết để xác định loại CSDL nào phù hợp với ứng dụng.

### Các loại nhất quán

Nhất quán (Consistency) có thể bao gồm các loại sau:

**Nhất quán với các người dùng khác**: nếu hai người dùng cùng truy cập CSDL vào một thời điểm, họ có thể cùng thấy chung một dữ liệu không. RDBMS sẽ cố gắng đảm bảo điều đó, trong khi CSDL NoSQL thường có những lựa chọn "thoải mái" hơn. Nhất quán trong một phiên làm việc Dữ liệu có duy trì một ràng buộc logic trong bối cảnh của một phiên làm việc với CSDL không? Ví dụ: nếu chúng ta cập nhật một hàng, và sau đó đọc lại nó, chúng ta có thể thấy các thay đổi đó hay không.

**Nhất quán trong một request:** có phải một request đơn lẻ là nhất quán cục bộ (internally coherent)? Ví dụ chúng ta đọc tất cả các hàng trong một bảng quan hệ, chúng ta muốn đảm bảo để trạng thái của bảng là trong chính thời điểm hiện tại. Sẽ không có bất cứ sự cập nhật dữ liệu nào có thể xay ra khi chúng ta bắt đầu câu truy vấn.

**Nhất quán với thực tại:** có phải dữ liệu chính xác trong thời điểm hiện tại luôn được tham chiếu tới? Ví dụ, trong một giao dịch của ngân hàng, số dư tài khoản phải đạt được nhất quán trong suốt khoảng thời gian giao dịch. Tức là phải đưa ra được số dư chính xác của tài khoản theo thời gian thực trong suốt thời gian giao dịch, kể cả khi có giao dịch khác xem vào giữa giao dịch làm làm thay đổi số dư của tài khoản.

### ACID và MVCC

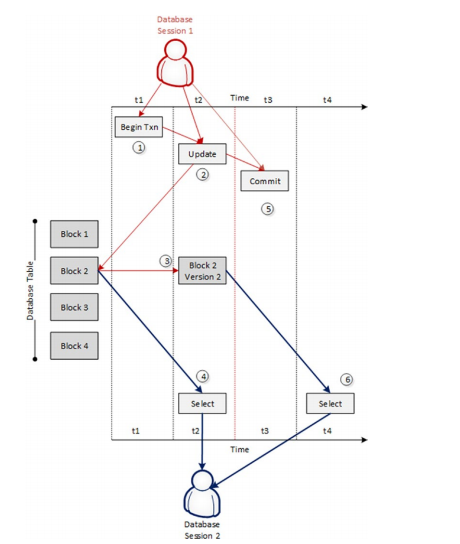
RDBMS yêu cầu sự nhất quán được thực hiện bằng cách sử dụng các mẫu kiến trúc: ACID transactions và multi-version concurrency control. Chúng ta đã nói đến ACID trước đây, chúng bao gồm Atomic, Consitent, Isolated và Durable:

* **Atomicity** (tính nguyên tố) là một giao dịch có nhiều hoạt động thì hoặc là toàn bộ hoạt động được thực hiện thành công hoặc không hoạt động nào được thực hiện.
* **Consistency** (tính nhất quán): một giao dịch hoặc là sẽ tạo ra trạng thái mới và hợp lệ cho dữ liệu, hoặc trong trường hợp có lỗi sẽ chuyển toàn bộ dữ liệu về trạng thái trước khi thực hiện giao dịch.
* **Isolation** (tính cô lập): một giao dịch đang được thực thi phải đảm bảo được tách biệt bởi giao dịch khác.
* **Durability** (tính bền vững): dữ liệu được xác nhận sẽ được hệ thống lưu lại sao cho ngay cả trong trường hợp hỏng móc hoặc có lỗi hệ thống, dữ liệu vẫn phải được đảm bảo trong tình trạng chuẩn xác.

Cách dễ nhất để cài đặt ACID là sử dụng lock. Tuy nhiên, sử dụng lock có thể dẫn đến sự tranh chấp cao khó chấp nận và khả năng song song thấp.

Để cung cấp ACID consistency mà không cần sử dụng thừa các kóa, RDBMS cung cấp mô hình MVVC. Trong mô hình này, nhiều bản sao chép của dữ liệu được gắn nhãn với mốc thời gian hoặc thay đổi định danh để cho phép CSDL xây dựng một snapshot của CSDL tại thời một thời điểm. Bằng cách này, MVCC có thể cung cấp cho giao dịch một sự cô lập và nhất quán trong khi vẫn tối đa hóa khả năng song song.

Ví dụ, trong MVVC, nếu một bảng dữ liệu đang được cập nhật trong khoảng thời gian bắt đầu đọc bảng cho đến khi kết thúc phiên làm việc, CSDL sẽ sử dụng phiên bản cũ của dữ liệu ở trong bảng để đảm bảo rằng dữ liệu thấy được là một phiên bản nhất quán của dữ liệu. Điều này cũng có nghĩa rằng, cho đến khi giao dịch được commit, các phiên làm việc khác sẽ không nhìn thấy sự thay đổi của giao dịch mà chỉ thấy phiên bản dữ liệu cũ hơn. Những dữ liệu này cũng dùng để rollback lại giao dịch nếu chúng thực hiện không thành công. Ưu điểm lớn nhất của MVVC là giảm quá tải lock.



Hình 2. Multi-version concurrency control (MVCC)

MVVC sử dụng timestamp để xác định phiên bản của dữ liệu. Tuy nhiên phần lớn các CSDL sử dụng global transaction ID thay thế cho timestamp. Trong Oracle, nó được gọi là system change number (SCN) và trong SQL Server được gọi là transaction sequence number.

Sequence number này tự động tăng khi một transaction được bắt đầu, và nó được lưu trữ trong cấu trúc thông tin của những hàng (rows, database blocks) đã được modified. Khi có một yêu cầu truy vấn, hệ QTCSDL sẽ tìm kiếm nhữngdòngdòng có sequence number nhỏ hơn hoặc bằng so với sequence number hiện tại (khi truy vấn bắt đầu được thực thi), khi bắt gặp một dòng có sequence number cao hơn, hệ QTCSDL sẽ hiểu rằng nó phải tìm kiếm một version cũ hơn của dòng đó.

### Two phase commit

MVVC hoạt động cùng với ACID transaction để dungung cấp sự cô lập giữa các giao dịch trong ùngùng một hệ thống. Giao dịch giữa các CSDL trong hệ thông RDBMS bằng cách sử dụng giao thức Two-phase commit.

Trong quá trình xác nhận, nếu một trong các nút của hệ thống phủ nhận giao dịch thì toàn bộ giao dịch sẽ bị hủy bỏ. Giao dịch chỉ được hoàn thành khi tất cả các nút của hệ thống xác nhận nó. Quá trình xác nhận bao gồm 2 pha là:

* Gửi yêu cầu: nút điều phối (coordinator) gửi câu truy vấn yêu cầu xác nhận giao dịch tới tất cả các nút tham gia, sau đó đợi trả lời từ các nút đó.
* Xác nhận giao dịch: khi nút điều phối nhận được trả lời đồng ý của tất cả các nút tham gia, nó sẽ gửi lại thông điệp xác nhận tới tất cả các nút tham gia một lần nữa để hoàn tất việc giao dịch, và giao dịch đó được xác nhận thành công. Trong trường hợp nút điều phối nhận được ít nhất một trả lời phủ định giao dịch từ một nút tham gia nào đó, hoặc nếu sau khoảng thời gian nhất định vẫn chưa nhận được toàn bộ trả lời từ các nút tham gia, thì nút điều phối sẽ gửi thông điệp hủy bỏ việc xác nhận tới tất cả các nút tham gia, và việc xác nhận giao dịch thất bại.

### Các mức độ nhất quán

Trong các CSDL NoSQL, gần như không có hệ thống nào cung cấp sự nhất quán dữ liệu giữa nhiều câu truy vấn: ví dụ ta có thể xóa một dòng trong một bảng và thêm một dòng khác vào bảng như là một atomic operation. Tuy vậy, đối với các single-object operation, các hệ CSDL NoSQL lại cung cấp nhiều các mức độ nhất quán mà chúng ta có thể mong đợi. Một số mức độ đáng chú ý là:

**Strict consistency**: Một yêu cầu đọc luôn trả về giá trị mới nhất.

**Causal consistency**: Yêu cầu đọc có thể không trả về giá trị gần nhất, tuy nhiên nó sẽ không trả về giá trị "out of sequence". Ví dụ một phiên làm việc được tạo ra với mục đích update A,B và C thì một phiên làm việc khác chỉ có thể thấy A,B,C cùng được update hoặc chưa.

**Monotonic consistency**: Một phiên làm việc sẽ không bao giờ có thể thấy được dữ liệu được phục hồi từ trước thời điểm nó bắt đầu. Một khi bắt đầu đọc dữ liệu, chúng ta sẽ không bao giờ thấy các phiên bản trước đây của dữ liệu so với thời điểm bắt đầu phiên làm việc.

**Eventual consistency**: Hệ thống có thể không nhất quán tại một thời điểm nào đó, nhưng mỗi operation độc lập cuối cùng đều sẽ được nhất quán. Nếu tất cả các thao tác cập nhật dữ liệu đều dùng lại, hệ thống cuối cùng cũng sẽ đạt được trạng thái nhất quán.

**Weak consistency**: Hệ thống sẽ không đảm bảo rằng hệ thống sẽ trở nên nhất quán, ví dụ, nếu lỗi server, một thao tác cập nhật dữ liệu có thể bị hỏng.

Thông thường, các hệ CSDL NoSQL đều cài đặt strict hoặc eventual consistency trong khi RDBMS cung cấp ACID consistency.

### Các cấp độ độc lập (Isolation)

Isolation level là một thuộc tính của transaction, qui định mức độ cô lập của dữ liệu mà transaction có thể truy nhập vào khi dữ liệu đó đang được cập bởi một transaction khác. Khi một transaction cập nhật dữ liệu đang diễn ra, một phần dữ liệu sẽ bị thay đổi (ví dụ một số bản ghi của bảng được sửa đổi hoặc bị xóa bỏ, một số được thêm mới), vậy các transaction hoặc truy vấn khác xảy ra đồng thời và cùng tác động vào các bản ghi đó sẽ diễn ra thế nào? Chúng sẽ phải đợi đến khi transaction đầu hoàn thành hay có thể thực hiện song song, kết quả dữ liệu nhận được là trong khi hay sau khi cập nhật? Bạn có thể điều khiển những

hành vi này thông qua việc đặt isolation level của từng transaction.

**Read Uncommited:** khi transaction thực hiện ở mức này, các truy vấn vẫn có thể truy nhập vào các bản ghi đang được cập nhật bởi một transaction khác và nhận được dữ liệu tại thời điểm đó mặc dù dữ liệu đó chưa được commit (uncommited data). Nếu vì lý do nào đó transaction ban đầu rollback lại những cập nhật, dữ liệu sẽ trở lại giá trị cũ. Khi đó transaction thứ hai nhận được dữ liệu sai (dirty read).

**Read commited:** transaction sẽ không đọc được dữ liệu đang được cập nhật mà phải đợi đến khi việc cập nhật thực hiện xong. Vì thế nó tránh được dirty read như ở mức trên.

Nó được cài đặt trong DBMS bằng lock-based concurrency control bằng cách khóa write locks (trên các dữ liệu đã được selected) cho đến khi kết thúc transaction, nhưng read locks được giải phóng ngay sau khi lệnh SELECT được thực thi xong. Vì vậy có thể xảy ra non-repeatable read.

**Repeatable read:** mức isolation này hoạt động nhứ mức read commited nhưng nâng thêm một nấc nữa bằng cách ngăn không cho transaction ghi vào dữ liệu đang được đọc bởi một transaction khác cho đến khi transaction khác đó hoàn tất.

Nó được cài đặt trong DBMS bằng cách giữ cả khóa write locks và read locks (trên các dữ liệu được select) cho đến khi kết thúc transaction, vì vậy không xảy ra non-repeatable read. Tuy nhiên có thể xảy ra trường hợp hai truy vấn trong cùng một transaction có thể trả về hai kết quả khác nhau (phantom read) do range locks không được quản lý.

**Serializable:** mức isolation này tăng thêm một cấp nữa và khóa toàn bộ dải các bản ghi có thể bị ảnh hưởng bởi một transaction khác, dù là UPDATE/DELETE bản ghi đã có hay INSERT bản ghi mới.

Nó được cài đặt bằng cách giữ cả khóa read locks và write locks cho đến khi kết thúc transaction. Range locks cũng được khóa cho khi có câu truy vấn SELECT sử dụng mệnh đề clause.

**Snapshot:** mức độ này cũng đảm bảo độ cô lập tương đương với Serializable, nhưng nó hơi khác ở phương thức hoạt động. Khi transaction đang select các bản ghi, nó không khóa các bản ghi này lại mà tạo một bản sao (snapshot) và select trên đó. Vì vậy các transaction khác insert/update lên các bản ghi đó không gây ảnh hưởng đến transaction ban đầu. Tác dụng của nó là giảm blocking giữa các transaction mà vẫn đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên cái giá kèm theo là cần thêm bộ nhớ để lưu bản sao của các bản ghi, và phần bộ nhớ này là cần cho mỗi transaction do đó có thể tăng lên rất lớn.

Các mức isolation từ 1 – 4 kể trên tăng theo thứ tự mức độ cô lập dữ liệu, giúp tăng tính toàn vẹn dữ liệu và nhất quán của transaction. Đồng thời nó cũng tăng thời gian chờ lẫn nhau của các transaction. Khi càng lên mức cao, đòi hỏi về tính toàn vẹn dữ liệu càng cao và càng có nhiều tình huống một transaction ngăn không cho các transaction khác truy nhập vào dữ liệu mà nó đang thao tác. Do đó nó càng tăng tình trạng locking và blocking trong database (ngoại trừ với snapshot thì tăng lượng bộ nhớ cần sử dụng). Hiệu năng của hệ thống do đó bị giảm đi. Thông thường, mức isolation read commited (mức mặc định) là phù hợp trong đa số các ứng dụng. Có thể một vài chức năng quan trọng (ví dụ chức năng ở trang admin update dữ liệu có ảnh hưởng đến toàn hệ thống) bạn cần tính toàn vẹn cao và phải chọn mức isolation cao hơn. Hoặc có những chức năng cần ưu tiên tốc độ thực hiện và có thể chấp nhận một chút dữ liệu không nhất quán, bạn có thể đặt xuống mức read uncommited. Bảng dưới đây tóm tắt các tính năng của từng mức isolation.

|  |  |  |  |
| --- | --- | --- | --- |
| Mức Isolation | Dirty read | Non-repeatable read | Phantom read |
| Read Uncommitted | Yes | Yes | Yes |
| Read Committed | No | Yes | Yes |
| Repeatable read | No | No | Yes |
| Serializable | No | No | No |
| Snapshot | No | No | No |

# Công nghệ sử dụng

## Epoll API trong linux

Để xây dựng mô hình client-server cho HQT CSDL theo dạng cột, phải sử dụng Inter-Process Communication (IPC) là kỹ thuật truyền thông giữa các tiến trình. Mà cụ thể ở đây là socket. Bản chất của socket trong hệ điều hành Linux là các file descriptors (fd), là một tài nguyên thực hiện I/O trong Linux.

Mỗi client kết nối tới server sẽ là một socket tương đương với một fd, vậy khi số lượng lớn client kết nối tới nhiều server thì ta phải quản lý số lượng lớn các fd. Epoll API ở đây được sử dụng để nhằm quản lý các fd.

### Khái niệm epoll API

Epoll là một API trong được cung cấp bởi nhân Linux, xuất hiện lần đầu trong phiên bản Linux 2.6. Epoll cho phép một tiến trình có thể giám sát nhiều fd để kiểm tra các yêu cầu I/O khả dụng trên các fd. Epoll cung cấp một hiệu năng tốt hơn khi giám sát một số lượng lớn các fd.

Cấu trúc dữ liệu trung tâm của epoll API là một *epoll instance*. *epoll instance* này sẽ thực được tham chiếu bởi một open fd. Nhưng fd này không sử dụng cho việc I/O. Thay vào đó, nó phục vụ cho việc điều khiển cấu trúc dữ liệu của *epoll instance* nằm trong kernel. *epoll instance* có 2 mục đích chính là: (i) lưu trữ một danh sách các fd mà tính tiến trình muốn giám sát khi có các sự kiện I/O gọi là *danh sách giám sát (interest list)*, (ii) duy trì một danh sách các fd mà xảy ra sự kiện I/O và đang đợi xử lý được gọi là *danh sách sẵn sàng (ready list)*.

### Các lời gọi hệ thống trong epoll API

Epoll API cung cấp 3 lời gọi hệ thống là: (i) *epoll\_create*, (ii) *epoll\_ctrl*, (iii) *epoll\_wait*.

**epoll\_create:** Trong đó lời gọi hệ thống *epoll\_create* tạo ra một *instance* *epoll* mới với danh sách và trả về một file descriptor tham chiếu tới instance đó.

#inlucde <sys/epoll.h>

int epoll\_create(int *size*)

Tham số đầu vào *size* chỉ ra số lượng fd được mong chờ sẽ giám sát trong *epoll instance*. Từ phiên bản Linux 2.6.8, tham số này đã bị bỏ qua. Hàm sẽ trả về fd tham chiếu tới *epoll instance* và khi không còn sử dụng nữa thì sử dụng hàm *close()* để đóng fd này lại.

**epoll\_ctl:** là lời gọi hệ thống dùng cho việc thay đổi *danh sách giám sát* trong *epoll instance*.

#include <sys/epoll.h>

int epoll\_ctl(int *epfd*, int *op*, int *fd*, struct epoll\_event *\*ev*);

Tham số *epfd* là giá trị trả về bởi *epoll\_create* trỏ tới *epoll\_instance* mà có *danh sách giám sát* nó quan tâm.

Tham số *fd* xác định file descriptor trong *danh sách giám sát* sẽ được thay đổi.

Tham số *op* xác định hành động sẽ được thực hiện trong *danh sách giám sát*, tham số này nhận một trong các giá trị sau:

* EPOLL\_CTL\_ADD: thêm file descriptor *fd* vào trong *danh sách giám sát* của *epfd*. Tập các sự kiện sẽ được giám sát được xác định ở trong bộ đệm trỏ bởi con trỏ *ev*. Nếu thực sự đã tồn tại file descriptor ở trong *danh sách giám sát*. Hàm sẽ trả về lỗi EEXIST.
* EPOLL\_CTL\_MOD: Thay đổi các sự kiện được giám sát của *fd* trong *danh sách giám sát*. Nếu *fd* mà không nằm trong *danh sách giám sát* thì hàm thất bại sẽ trả về lỗi ENOENT.
* EPOLL\_CTL\_DEL: Xóa *fd* khỏi danh sách giám sát ở trong *epfd*. Tham số *ev* sẽ được bỏ qua đối với hành động này. Nếu *fd* không nằm trong *danh sách giám sát* thì hàm sẽ thất bại và trả về lỗi ENOENT. Việc đóng file descriptor sẽ tự động xóa file đó khỏi *danh sách giám sát* của tất cả các *epoll­\_instance* mà nó là thành viên.

Tham số *ev* là một con trỏ mà trỏ tới cấu trúc *epoll\_event*, được định nghĩa như sau:

struct epoll\_event {

uint32\_t events; /\* các sự kiện của epoll \*/

epoll\_data\_t data; /\* dữ liệu người dùng \*/

}

Trường *data* trong cấu trúc *epoll\_event,* được định nghĩa như sau:

typedef union epoll\_data {

void \*ptr; /\* Con trỏ mà tới dữ liệu của người dùng \*/

int fd; /\* File descriptor \*/

uint32\_t u32; /\* 32 bit integer \*/

uint64\_t u64; /\* 64 bit integer \*/

} epoll\_data\_t;

Tham số *ev* xác định

Trường con *events* là một bit mask xác định tập các sự kiện mà chúng ta quan tâm trong giám sát *fd.* Các cơ thông dụng được sử dụng là: (i) EPOLLIN quan tâm tới các sự kiện INPUT, (ii) EPOLLOUT quan tâm tới các sự kiện OUTPUT, (iii) EPOLLERR quan tâm tới các lỗi xảy ra, (iv) EPOLLET sử dụng thông báo sự kiện edge-triggered.

Trường con *data* là một union, dùng để xác định thông tin được truyền trở lại tới hàm được gọi nếu *fd* nằm trong *danh sách sẵn sàng*.

**epoll\_wait:** lời gọi hệ thống này sẽ trả về thông tin về các file descriptor nằm trong *danh sách sẵn sàng* từ *epoll instance*. *epoll\_wait* có thể trả về nhiều file descriptor sẵn sàng.

#include <sys/epoll.h>

int epoll\_wait(int *epfd*, struct epoll\_event *\*evlist*, int *maxevents*,

int *timeout*);

Tham số *epfd* tham chiếu tới *epoll\_instance* mà chúng ta quan tâm.

Thông tin về các fd trong *danh sách sẵn sàng* được trả về trong mảng cấu trúc *epoll\_event* trỏ bởi *evlist*. Mảng *evlist* sẽ được cấp phát bởi người gọi, và số lượng các thành phần sẽ được chỉ ra trong tham số *maxevents.*

Trường con *events* sẽ trả về một mask các sự kiện mà xảy ra trong file descriptor đó. Trường con *data* trả về bất cứ giá trị nào được xác định trong *ev.data* khi nó được thêm vào.

Tham số *timeout* xác định thời gian đợi cho đến khi có phần tử trong *danh sách sẵn sàng*.

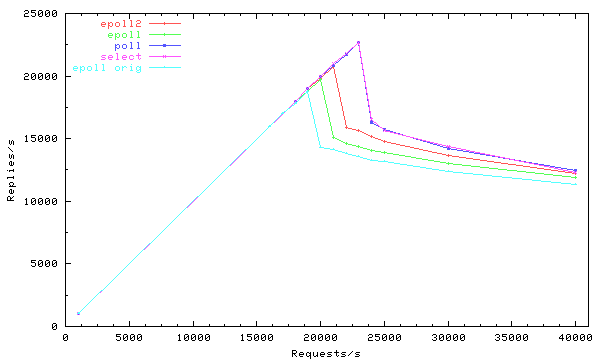
* Nếu *timeout* bằng -1, hàm sẽ đợi cho đến khi có phần tử nằm trong *danh sách sẵn sàng*.
* Nếu *timeout* bằng 0, thực hiện kiểm tra *danh sách sẵn sàng* và trả về ngay lập tức kể cả khi không có phần tử nào trong danh sách.
* Nếu *timeout* lớn hơn 0, sẽ thực hiện đợi trong *timeout* micro giây hoặc cho đến khi có phần tử trong *danh sách sẵn sàng*.

Nếu thành công *epoll\_wait* sẽ trả về số lượng phần tử đã được đặt trong mảng *evlist* hoặc 0 nếu không có file descriptor trong *danh sách sẵn sàng*. Nếu lỗi, *epoll\_wait()* trả về -1.

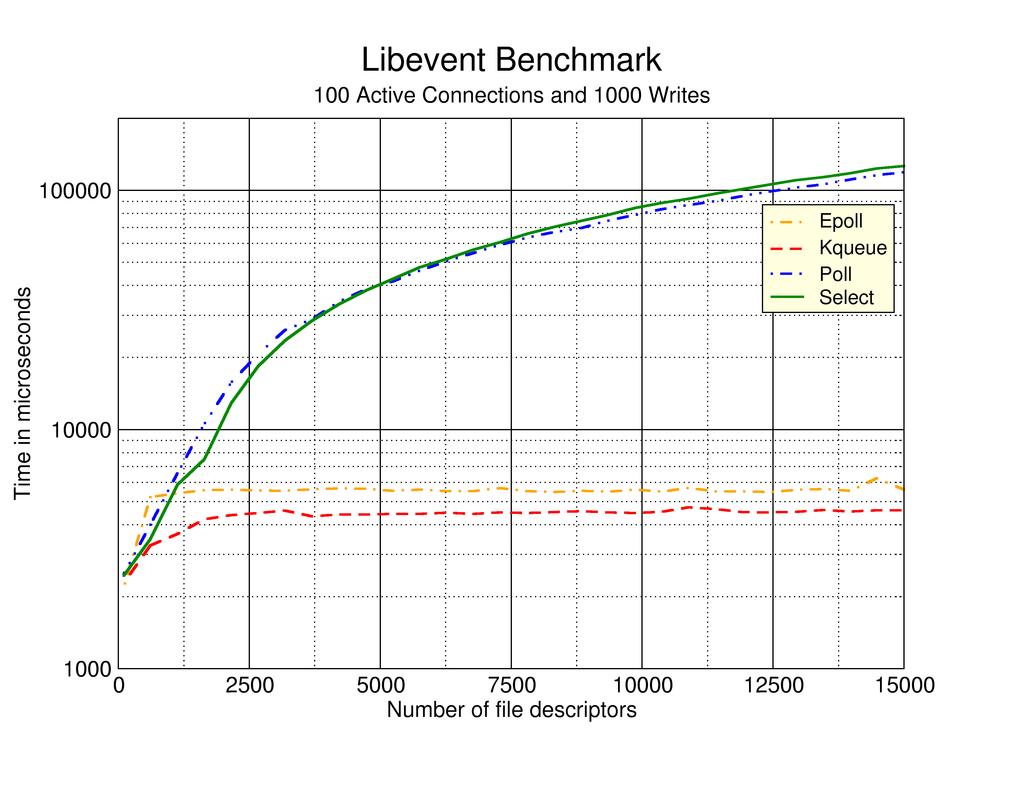
### Ưu điểm và nhược điểm của epoll

**Ưu điểm:**

* Đạt được hiệu năng cao khi giám sát một số lượng lớn các fd
* Xác định rõ ràng file descriptor mà chúng ta muốn giám sát
* Có thể lựa chọn giữa thông báo edge-triggred và thông báo level-triggered



Hình 3. So sánh hiệu năng giữa epoll với select, poll



Hình 4. So sánh hiệu năng epol với poll, select và kqueue trong thư viện libevent

**Nhược điểm:**

* Nó là một API của Linux

## Bộ thư viện glibc

Đối với CSDL được xây dựng ở trong đề tài, dữ liệu sẽ được lưu trữ ở trong file với cấu trúc được thiết kế sẽ được chỉ ra trong Chương 4. Để thực hiện các công việc vào ra trong trong file, chương trình sẽ tập trung sử dụng các hàm tron g thư viện glibc của Linux. Cụ thể đó là các hàm *open, close, read, write, lseek*.

### open

Hàm open có nguyên mẫu:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int open(const char \**path\_name*, int *flags*, mode\_t *mode*);

*open* là một lời gọi hệ thống dùng để mở file xác định trong *path\_name*. Nếu file không tồn tại, nó có thể tạo ra một file mới nếu cờ O\_CREAT­ được xác định trong tham số *flags*. Giá trị trả về của hàm là một fd nếu thành công, và fd này sẽ dùng để thực hiện I/O với file.

Tham số *flags* phải bao gồm một trong các cờ sau: (i) O\_RDONLY chỉ đọc file, (ii) O\_WRONLY chỉ ghi file, (iii) O\_RDWR thực hiện cả đọc và ghi file.

Tham số *mode* xác định các quyền của file được áp dụng khi file mới được tạo. Tham số này sẽ được dùng khi tham số *flags* có cờ O\_CREAT hoặc O\_TMPFILE. Nếu ­*flags* không có 1 trong 2 cờ này thì tham số *mode* sẽ bị bỏ qua. Tham số *mode* có các cờ sau: (i) S\_IRWXU người sở hữu có quyền đọc, ghi và thực thi, (ii) S\_IRUSR người sở hữu có quyền đọc, (iii) S\_ IWUSR người sở hữu có quyền ghi, (iv) S\_IXUSR người sở hữu có quyền thực thi, (v) S\_IRWXG nhóm của người sở hữu có quyền đọc, ghi và thực thi, (vi) S\_IWGRP nhóm của người sở hữu có quyền ghi, (vii) S\_IXGRP nhóm của người sở hữu có quyền thực thi, (viii) S\_IRWXO tất cả người dùng có quyền đọc, ghi và thực thi, (ix) S\_IROTH tất cả người dùng đều có quyền đọc, (x) S\_IWOTH tất cả người dùng đều có quyền ghi, (xi) S\_IXOTH tất cả người dùng đều có quyền thực thi.

### close

Hàm close có nguyên mẫu:

#inlcude <unistd.h>

int close(int *fd*);

Hàm close sẽ thực hiện đóng file descriptor lại và nếu *fd* là tham chiếu cuối cùng tới file descriptor thì các tài nguyên cấp phát cho file descriptor sẽ được giải phóng.

### read

Hàm read có nguyên mẫu:

#include <unistd.h>

ssize\_t read(int *fd*, void \**buf*, size\_t *count*);

Hàm read sẽ thực hiện đọc *count* byte từ *fd* vào trong buffer mà bắt đầu từ *buf*.

### write

Hàm write có nguyên mẫu:

#include <unistd.h>

ssize\_t write(int *fd*, const void \**buf*, size\_t *count*);

Hàm write thực hiện viết *count* byte từ buffer được trỏ bởi *buf* tới file được tham chiếu bởi file descriptor *fd.*

### lseek

Hàm lseek có nguyên mẫu

#include <sys/types.h>

#include <unistd.h>

Off\_t lseek(int *fd*, off\_t *off\_set*, int *whence*);

Hàm lseek thực hiện việc định lại vị trí của file được tham chiếu bởi *fd*. Vị trí mới sẽ được xác định cách *offset* byte từ *whence*. *whence* có thể nhận các giá trị: (i) SEEK\_SET là vị trí đầu của file, (ii) SEEK\_CUR là vị trí hiện tại của file, (iii) SEEK\_END vị trí cuối cùng của file.

## Hàm băm double hashing đề xuất bởi Donald Knuth

### Double hashing

# Phát triển và triển khai ứng dụng

## Thiết kế kiến trúc

### Cấu trúc trong thông tin cơ sở dữ liệu

### Cấu trúc trong thông tin bảng

### Cấu trúc lưu trữ dữ liệu

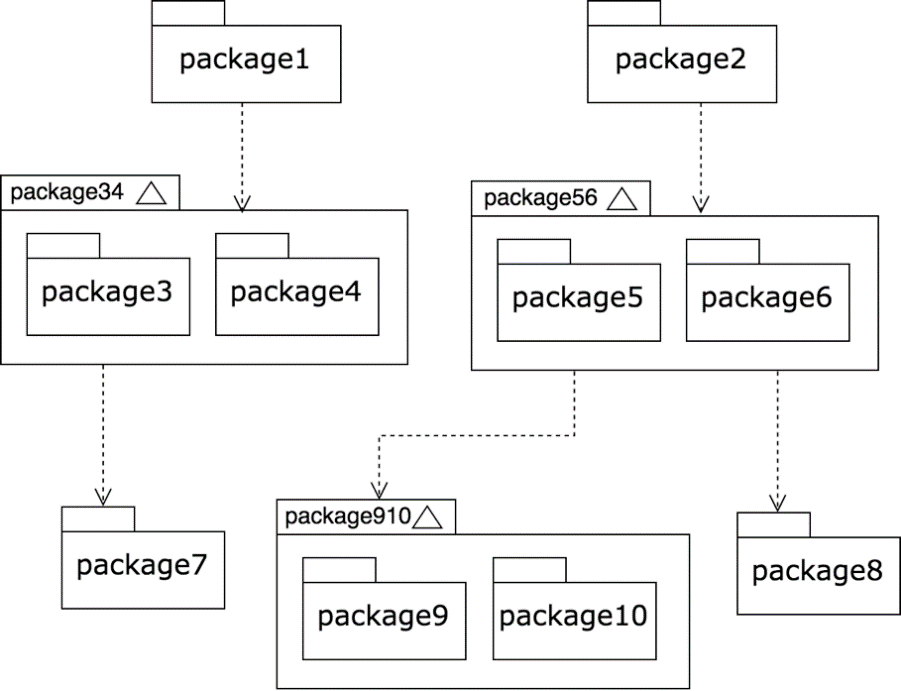
### Lựa chọn kiến trúc phần mềm

Mục này có độ dài từ một đến ba trang. Sinh viên cần lựa chọn kiến trúc phần mềm cho ứng dụng của mình như: kiến trúc ba lớp MVC, MVP, SOA, Microservice, v.v. rồi giải thích sơ bộ về kiến trúc đó (không giải thích chi tiết/dài dòng).

Sử dụng kiến trúc phần mềm đã chọn ở trên, sinh viên mô tả kiến trúc cụ thể cho ứng dụng của mình. Gợi ý: sinh viên áp dụng lý thuyết chung vào hệ thống/sản phẩm của mình như thế nào, có thay đổi, bổ sung hoặc cải tiến gì không. Ví dụ, thành phần M trong kiến trúc lý thuyết MVC sẽ là *những thành phần cụ thể nào* (ví dụ: là interface I + class C1 + class C2, v.v.) trong kiến trúc phần mềm của sinh viên.

### Thiết kế tổng quan

Sinh viên vẽ biểu đồ gói UML (UML package diagram), nêu rõ sự phụ thuộc giữa các gói (package). SV cần vẽ các gói sao cho chúng được phân theo các tầng rõ ràng, không được sắp đặt package lộn xộn trong hình vẽ. Sinh viên chú ý các quy tắc thiết kế (Các gói không phụ thuộc lẫn nhau, gói tầng dưới không phụ thuộc gói tầng trên, không phụ thuộc bỏ qua tầng, v.v.) và cần giải thích sơ lược về mục đích/nhiệm vụ của từng package. SV tham khảo ví dụ minh họa trong Hình 1.



**Hình 5** Ví dụ biểu đồ phụ thuộc gói

### Thiết kế chi tiết gói

Sinh viên thiết kế và lần lượt vẽ biểu đồ thiết kế cho từng package, hoặc một nhóm các package liên quan để giải quyết một vấn đề gì đó. Khi vẽ thiết kế gói, sinh viên chỉ cần đưa tên lớp, không cần chỉ ra các thành viên phương thức và thuộc tính. SV tham khảo ví dụ minh họa trong Hình 2.

Sinh viên cần vẽ rõ ràng quan hệ giữa các lớp trong biểu đồ. Các quan hệ bao gồm: phụ thuộc (dependency), kết hợp (association), kết tập (aggregation), hợp thành (composition), kế thừa (inheritance), và thực thi (implementation). Các quan hệ này đều đã được minh họa trong Hình 2.

Sau khi vẽ hình minh họa, sinh viên cần giải thích ngắn gọn về thiết kế của mình.



**Hình 6** Ví dụ thiết kế gói

## Thiết kế chi tiết

### Thiết kế giao diện

Phần này có độ dài từ hai đến ba trang. Sinh viên đặc tả thông tin về màn hình mà ứng dụng của mình hướng tới, bao gồm độ phân giải màn hình, kích thước màn hình, số lượng màu sắc hỗ trợ, v.v. Tiếp đến, sinh viên đưa ra các thống nhất/chuẩn hóa của mình khi thiết kế giao diện như thiết kế nút, điều khiển, vị trí hiển thị thông điệp phản hồi, phối màu, v.v. Sau cùng sinh viên đưa ra một số hình ảnh minh họa thiết kế giao diện cho các chức năng quan trọng nhất. Lưu ý, sinh viên không nhầm lẫn giao diện thiết kế với giao diện của sản phẩm sau cùng.

### Thiết kế lớp

Phần này có độ dài từ ba đến bốn trang. Sinh viên trình bày thiết kế chi tiết các thuộc tính và phương thức cho một số lớp chủ đạo/quan trọng nhất của ứng dụng (từ 2-4 lớp). Thiết kế chi tiết cho các lớp khác, nếu muốn trình bày, sinh viên đưa vào phần phụ lục.

Để minh họa thiết kế lớp, sinh viên thiết kế luồng truyền thông điệp giữa các đối tượng tham gia cho 2 đến 3 use case quan trọng nào đó bằng biểu đồ trình tự (hoặc biểu đồ giao tiếp).

### Thiết kế cơ sở dữ liệu

Phần này có độ dài từ hai đến bốn trang. Sinh viên thiết kế, vẽ và giải thích biểu đồ thực thể liên kết (E-R diagram). Từ đó, sinh viên thiết kế CSDL tùy theo hệ quản trị CSDL mà mình sử dụng (SQL, NoSQL, Firebase, v.v.)

## Xây dựng ứng dụng

### Thư viện và công cụ sử dụng

Sinh viên liệt kê các công cụ, ngôn ngữ lập trình, API, thư viện, IDE, công cụ kiểm thử, v.v. mà mình sử dụng để phát triển ứng dụng. Mỗi công cụ phải được chỉ rõ phiên bản sử dụng. SV nên kẻ bảng mô tả tương tự như Bảng 1. Nếu có nhiều nội dung trình bày, sinh viên cần xoay ngang bảng.

**Bảng 1** Danh sách thư viện và công cụ sử dụng

|  |  |  |
| --- | --- | --- |
| **Mục đích** | **Công cụ** | **Địa chỉ URL** |
| IDE lập trình | Eclipse Oxygen 64 bit | http://www.eclipse.org/ |
| v.v. | v.v. | v.v. |

### Kết quả đạt được

Sinh viên trước tiên mô tả kết quả đạt được của mình là gì, ví dụ như các sản phẩm được đóng gói là gì, bao gồm những thành phần nào, ý nghĩa, vai trò?

Sinh viên cần thống kê các thông tin về ứng dụng của mình như: số dòng code, số lớp, số gói, dung lượng toàn bộ mã nguồn, dung lượng của từng sản phẩm đóng gói, v.v. Tương tự như phần liệt kê về công cụ sử dụng, sinh viên cũng nên dùng bảng để mô tả phần thông tin thống kê này.

### Minh hoạ các chức năng chính

Sinh viên lựa chọn và đưa ra màn hình cho các chức năng chính, quan trọng, và thú vị nhất. Mỗi giao diện cần phải có lời giải thích ngắn gọn. Khi giải thích, sinh viên có thể kết hợp với các chú thích ở trong hình ảnh giao diện.

## Kiểm thử

Phần này có độ dài từ hai đến ba trang. Sinh viên thiết kế các trường hợp kiểm thử cho hai đến ba chức năng quan trọng nhất. Sinh viên cần chỉ rõ các kỹ thuật kiểm thử đã sử dụng. Chi tiết các trường hợp kiểm thử khác, nếu muốn trình bày, sinh viên đưa vào phần phụ lục.

Sinh viên sau cùng tổng kết về số lượng các trường hợp kiểm thử và kết quả kiểm thử. Sinh viên cần phân tích lý do nếu kết quả kiểm thử không đạt.

## Triển khai

Sinh viên trình bày mô hình và/hoặc cách thức triển khai thử nghiệm/thực tế. Ứng dụng của sinh viên được triển khai trên server/thiết bị gì, cấu hình như thế nào. Kết quả triển khai thử nghiệm nếu có (số lượng người dùng, số lượng truy cập, thời gian phản hồi, phản hồi người dùng, khả năng chịu tải, các thống kê, v.v.)

# Các giải pháp và đóng góp nổi bật

Chương này có độ dài tối thiểu 5 trang, tối đa không giới hạn. Sinh viên cần trình bày tất cả những nội dung đóng góp mà mình thấy tâm đắc nhất trong suốt quá trình làm ĐATN. Đó có thể là một loạt các vấn đề khó khăn mà sinh viên đã từng bước giải quyết được, là giải thuật cho một bài toán cụ thể, là giải pháp tổng quát cho một lớp bài toán, hoặc là mô hình/kiến trúc hữu hiệu nào đó được sinh viên thiết kế.

Chương này là **cơ sở quan trọng** để các thầy cô đánh giá sinh viên. Vì vậy, sinh viên cần phát huy tính sáng tạo, khả năng phân tích, phản biện, lập luận, tổng quát hóa vấn đề và tập trung viết cho thật tốt.

Mỗi giải pháp hoặc đóng góp của sinh viên cần được trình bày trong một mục độc lập bao gồm ba mục con: (i) dẫn dắt/giới thiệu về bài toán/vấn đề, (ii) giải pháp, và (iii) kết quả đạt được (nếu có).

Sinh viên lưu ý **không trình bày lặp lại nội dung**. Những nội dung đã trình bày chi tiết trong các chương trước không được trình bày lại trong chương này. Vì vậy, với nội dung hay, mang tính đóng góp/giải pháp, sinh viên chỉ nên tóm lược/mô tả sơ bộ trong các chương trước, đồng thời tạo tham chiếu chéo tới đề mục tương ứng trong Chương 5 này. Chi tiết thông tin về đóng góp/giải pháp được trình bày trong mục đó.

Ví dụ, trong Chương 4, sinh viên có thiết kế được kiến trúc đáng lưu ý gì đó, là sự kết hợp của các kiến trúc MVC, MVP, SOA, v.v. Khi đó, sinh viên sẽ chỉ mô tả ngắn gọn kiến trúc đó ở Chương 4, rồi thêm các câu có dạng: “Chi tiết về kiến trúc này sẽ được trình bày trong phần 5.1”.

# Kết luận và hướng phát triển

## Kết luận

Sinh viên so sánh kết quả nghiên cứu hoặc sản phẩm của mình với các nghiên cứu hoặc sản phẩm tương tự.

Sinh viên phân tích trong suốt quá trình thực hiện ĐATN, mình đã làm được gì, chưa làm được gì, các đóng góp nổi bật là gì, và tổng hợp những bài học kinh nghiệm rút ra nếu có.

## Hướng phát triển

Trong phần này, sinh viên trình bày định hướng công việc trong tương lai để hoàn thiện sản phẩm hoặc nghiên cứu của mình.

Trước tiên, sinh viên trình bày các công việc cần thiết để hoàn thiện các chức năng/nhiệm vụ đã làm. Sau đó sinh viên phân tích các hướng đi mới cho phép cải thiện và nâng cấp các chức năng/nhiệm vụ đã làm.

# Tài liệu tham khảo

Lưu ý: Sinh viên không được đưa bài giảng/slide, các trang Wikipedia, hoặc các trang web thông thường làm tài liệu tham khảo.

Một trang web được phép dùng làm tài liệu tham khảo **chỉ khi** nó là công bố chính thống của cá nhân hoặc tổ chức nào đó. Ví dụ, trang web đặc tả ngôn ngữ XML của tổ chức W3C https://www.w3.org/TR/2008/REC-xml-20081126/ là TLTK hợp lệ.

Có năm loại tài liệu tham khảo mà sinh viên phải tuân thủ đúng quy định về cách thức liệt kê thông tin như sau. Lưu ý: các phần văn bản trong cặp dấu < > dưới đây chỉ là hướng dẫn khai báo cho từng loại tài liệu tham khảo; sinh viên cần xóa các phần văn bản này trong ĐATN của mình.

<**Bài báo đăng trên tạp chí khoa học**: Tên tác giả, tên bài báo, tên tạp chí, volume, từ trang đến trang (nếu có), nhà xuất bản, năm xuất bản>

1. Hovy E. H., Automated Discourse Generation Using Discourse Structure Relations, Artificial Intelligence, Elsevier Science Publishers, 63: 341-385, 1993.

<**Sách**: Tên tác giả, tên sách, volume (nếu có), lần tái bản (nếu có), nhà xuất bản, năm xuất bản>

1. Peterson L. L. and Davie B. S., Computer Networks: A Systems Approach, 2nd ed., Mogran-Kaufmann, 1999.
2. Nguyễn Thúc Hải, Mạng máy tính và các hệ thống mở, Nhà xuất bản giáo dục, 1999.

<**Tập san Báo cáo Hội nghị Khoa học**: Tên tác giả, tên báo cáo, tên hội nghị, ngày (nếu có), địa điểm hội nghị, năm xuất bản>

1. Poesio M. and Di Eugenio B., Discourse Structure and Anaphoric Accessibility, In Proc. of the ESSLLI Workshop on Information Structure, Discourse Structure and Discourse Semantics, Helsinki, 2001.

<**Đồ án tốt nghiệp, Luận văn Thạc sĩ, Tiến sĩ**: Tên tác giả, tên đồ án/luận văn, loại đồ án/luận văn, tên trường, địa điểm, năm xuất bản>

1. Knott D., A Data-Driven Methodology for Motivating a Set of Coherence Relations, Ph.D. Thesis, University of Edinburgh, UK, 1996.

<**Tài liệu tham khảo từ Internet**: Tên tác giả (nếu có), tựa đề, cơ quan (nếu có), địa chỉ trang web, thời gian lần cuối truy cập trang web>

1. Berners-Lee T., Hypertext Transfer Protocol (HTTP), CERN, ftp:/info.cern.ch/pub/www/doc/http-spec.txt.Z, last visited May 2010.
2. Princeton University, WordNet, http://www.cogsci.princeton.edu/~wn/index.shtml, last visited May 2010.

# Phụ lục

Phần phụ lục là không bắt buộc. Nếu sinh viên không có nhu cầu trình bày thêm, có thể xóa bỏ phần này. Lưu ý là phần phụ lục chỉ được đánh chỉ mục đến cấp 2, sinh viên không được phép chia nhỏ hơn nữa.

Hướng dẫn viết đồ án tốt nghiệp

Quy định chung

Dưới đây là một số quy định và hướng dẫn viết đồ án tốt nghiệp mà bắt buộc sinh viên phải **đọc kỹ và tuân thủ nghiêm ngặt**.

Sinh viên cần đảm bảo tính thống nhất toàn báo cáo (font chữ, căn dòng hai bên, hình ảnh, bảng, margin trang, đánh số trang, v.v.). Để làm được như vậy, sinh viên chỉ cần sử dụng các định dạng theo đúng template ĐATN này. Khi paste nội dung văn bản từ tài liệu khác của mình, sinh viên cần chọn kiểu Copy là “Text Only” để định dạng văn bản của template không bị phá vỡ/vi phạm.

Tuyệt đối cấm sinh viên đạo văn. Sinh viên cần ghi rõ nguồn cho tất cả những gì không tự mình viết/vẽ lên, bao gồm các câu trích dẫn, các hình ảnh, bảng biểu, v.v. Khi bị phát hiện, sinh viên sẽ **không được phép bảo vệ ĐATN**.

Tất cả các hình vẽ, bảng biểu, công thức, và tài liệu tham khảo trong ĐATN nhất thiết phải được SV giải thích và tham chiếu tới ít nhất một lần. Không chấp nhận các trường hợp sinh viên đưa ra hình ảnh, bảng biểu tùy hứng và không có lời mô tả/giải thích nào.

Sinh viên tuyệt đối không trình bày ĐATN theo kiểu viết ý hoặc gạch đầu dòng. ĐATN không phải là một slide thuyết trình; khi người đọc không hiểu sẽ không có ai giải thích hộ. Sinh viên cần viết thành các đoạn văn và phân tích, diễn giải đầy đủ, rõ ràng. Câu văn cần đúng ngữ pháp, đầy đủ chủ ngữ, vị ngữ và các thành phần câu.

Khi thực sự cần liệt kê, sinh viên nên liệt kê theo phong cách khoa học với các ký tự La Mã. Ví dụ, nhiều sinh viên luôn cảm thấy hối hận vì (i) chưa cố gắng hết mình, (ii) chưa sắp xếp thời gian học/chơi một cách hợp lý, (iii) chưa tìm được người yêu để chia sẻ quãng đời sinh viên vất vả, và (iv) viết ĐATN một cách cẩu thả.

Trong một số trường hợp nhất thiết phải dùng các bullet để liệt kê, sinh viên cần thống nhất Style cho toàn bộ các bullet các cấp mà mình sử dụng đến trong báo cáo. Nếu dùng bullet cấp 1 là hình tròn đen, toàn bộ báo cáo cần thống nhất cách dùng như vậy; ví dụ như sau:

* Đây là mục 1 – Thực sự không còn cách nào khác tôi mới dùng đến việc bullet trong báo cáo.
* Đây là mục 2 – Nghĩ lại thì tôi có thể không cần dùng bullet cũng được. Nên tôi sẽ xóa bullet và tổ chức lại hai mục này trong báo cáo của mình cho khoa học hơn. Tôi muốn thầy cô và người đọc cảm nhận được tâm huyết của tôi trong từng trang báo cáo ĐATN.

Tạo đề mục

Đề mục giúp tạo bố cục cho tài liệu. Để các tính năng tự động – ví dụ tính năng cập nhật mục lục, hoặc tính năng tham chiếu chéo – của Word hoạt động được, sinh viên cần tuân thủ theo các style đã tạo trong tài liệu này. Để hiển thị các style này, sinh viên vào tab Home trong thanh Ribbon của Word.

Để tạo đề mục cấp 1, 2, 3, 4, 5, sinh viên gõ tiêu đề cho đề mục của mình rồi chọn các style là Heading 1, 2, 3, 4, 5 tương ứng. Sinh viên hạn chế dùng tới đề mục cấp 4, và phải trong trường hợp thực sự cần thiết mới dùng đến đề mục cấp 5.

Phần phụ lục chỉ cho phép có hai cấp tiêu đề. Hai style tương ứng với hai cấp này là “Heading 7, Phụ lục cấp 1” và “Heading 8, Phụ lục cấp 2”.

Bảng biểu

Sinh viên lưu ý không để bảng tràn ra lề (margin) trên, dưới, trái hoặc phải của trang. Do không gian nhỏ hẹp, bảng nên có font là 12pt, nhỏ hơn một chút so với font thông thường (13pt) của ĐATN. Độ dãn dòng của bảng nên là 1 line. Căn lề bảng là căn giữa, nhưng nội dung văn bản trong bảng nên được căn lề trái.

Sinh viên có thể viết tắt các từ trong bảng để tiết kiệm không gian nhưng phải giải thích các từ viết tắt này ở phần Chú thích bảng. Ví dụ áp dụng được minh họa trong Bảng 2.

**Bảng 2** Ví dụ sử dụng bảng

Chú thích: Y: Year; RS: Risk Set; G: Graduated; AB: Academically Excluded;C: Censored; HRG: Hazard Ratio – Graduated

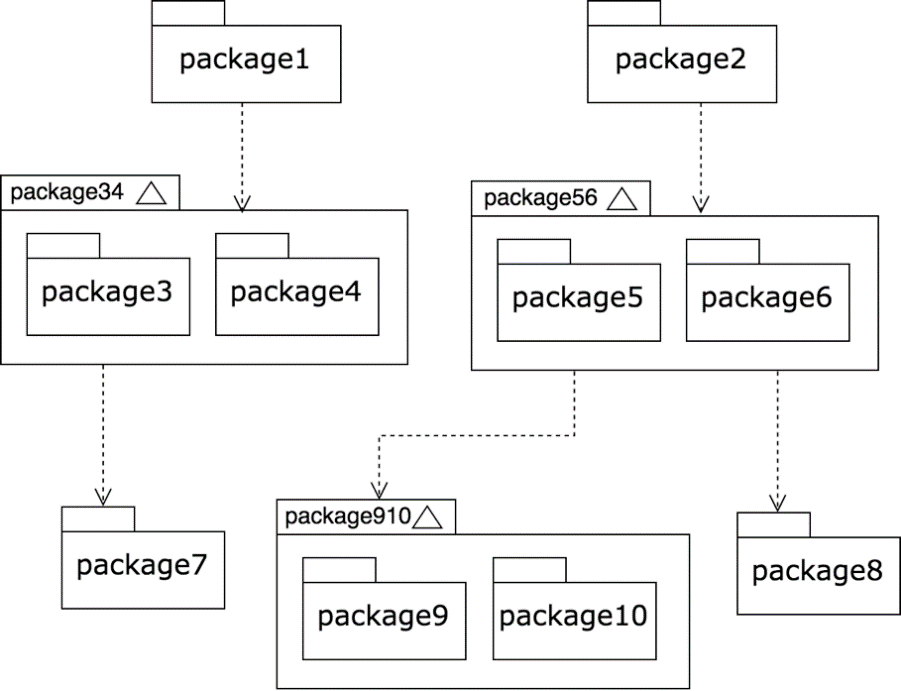
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Y** | **RS** | **G** | **AE** | **C** | **HRG**  **(%)** |
| **1** | 11.959 | 0 | 725 | 1619 | 0,0 |
| **2** | 10.457 | 0 | 474 | 1513 | 0,0 |
| **3** | 7.365 | 1213 | 335 | 966 | 16,7 |
| **4** | 900 | 599 | 145 | 405 | 55,3 |

Sinh viên được tùy ý lựa chọn định dạng (template) cho các bảng trong ĐATN của mình, nhưng phải đảm bảo tính nhất quán trên toàn ĐATN. Template bảng phải đảm bảo phần heading cột trong bảng có font in đậm và nổi bật (highlighted) hơn so với các nội dung khác trong bảng.

Vì bảng có thể kéo dài nhiều trang, tiêu đề của bảng nên để ở phần đầu của bảng. Sinh viên không thêm tiêu đề bảng bằng tay. Để thêm tiêu đề bảng tự động, sinh viên nhấn chuột phải vào bảng, chọn “Insert Caption”, chọn “Label” là “Bảng”, rồi nhấn nút “OK”. Sau đó, sinh viên nhập vào nội dung tiêu đề và căn chỉnh “Center” cho tiêu đề này. Lưu ý, sinh viên cần bôi đậm bằng tay cụm từ **Bảng n**. Kết quả thu được có dạng như sau “**Bảng 2** Ví dụ sử dụng bảng”. Sinh viên nên xoay ngang trang giấy trong trường hợp bảng có nhiều cột với nhiều nội dung văn bản.

Hình vẽ

Tương tự như bảng, sinh viên không được để hình vẽ tràn lề trang. Căn lề cho hình vẽ là căn giữa (Center). Cách thêm tiêu đề hình vẽ tương tự như cách thêm tiêu đề bảng, nhưng sinh viên chọn “Label” là “Hình” thay vì “Bảng”.



**Hình 7** Ví dụ hình vẽ

Tiêu đề hình vẽ phải đặt ở dưới hình vẽ. Nếu hình vẽ được copy từ trên mạng, sinh viên bắt buộc phải ghi rõ nguồn. Sinh viên nên thống nhất công cụ sử dụng và style cho hình vẽ trong toàn ĐATN. Các chi tiết trong hình vẽ phải được bố trí gọn gàng; chữ trong hình phải đảm bảo nhìn được rõ nét khi in báo cáo trên giấy A4. Khi resize ảnh, cần giữ nguyên tỷ lệ dài rộng, tránh làm méo hoặc vỡ hình. Ví dụ hình vẽ được minh họa trong Hình 3.

Tài liệu tham khảo

Sinh viên cần hạn chế tối đa dùng trang Web làm tài liệu tham khảo. Chỉ chấp nhận trang Web làm TLTK khi trang đó là nơi công bố chính thức của tổ chức hoặc cá nhân nào đó. Ví dụ, trang web đặc tả ngôn ngữ XML của tổ chức W3C https://www.w3.org/TR/2008/REC-xml-20081126/ là TLTK hợp lệ.

Khi giới thiệu về công cụ, API, thư viện, hoặc nền tảng nào đó, sinh viên có thể đưa ra địa chỉ URL của các tiện ích này. Sinh viên lưu ý địa chỉ URL đó không phải là tài liệu tham khảo. Trong các trường hợp tương tự như vậy, sinh viên nên tạo “Footnote”. Sinh viên tạo “Footnote” bằng cách vào mục “References”, chọn “Insert Footnote”. Ví dụ tạo Footnote như sau: TensorFlow[[1]](#footnote-1) là nền tảng học máy mã nguồn mở đang được sử dụng rộng rãi hiện nay. Lưu ý: số Footnote phải đặt sát với từ được mô tả. Như trong ví dụ trên, số 1 được đặt ngay cạnh chữ TensorFlow (không có dấu cách).

Công thức toán học

Giống như bảng, hình vẽ, và tài liệu tham khảo, công thức toán học cần được đánh số, giải thích, và tham chiếu đầy đủ.

Để thêm tiêu đề (caption) cho công thức, sinh viên đặt con trỏ văn bản vào dòng văn bản dưới công thức. Sau đó vào menu “Insert” (lưu ý là menu Insert, không phải là tab Insert trong thanh Ribbon) và chọn mục “Caption”. Từ Popup Menu hiện ra, sinh viên chọn “Label” là “Công thức”, rồi nhấn nút “OK”. Sinh viên bôi đậm chữ **Công thức n**, và căn tiêu đề ra giữa. Công thức 1 là ví dụ mẫu cho sinh viên tham khảo.

**Công thức 1** Khai triển Newton

Tham chiếu chéo

Tham chiếu chéo (Cross-reference) là tiện ích hữu hiệu cho người viết báo cáo. Nó giúp tạo các liên kết tham chiếu (hyperlink) tới các hình ảnh, bảng biểu, tài liệu tham khảo, và các đề mục một cách tự động. Ví dụ, ngay trong câu này, một tham chiếu đã được tạo ra tới mục 2.1. Người đọc dễ dàng nhấp chuột vào liên kết 2.1 để ngay lập tức chuyển đến mục đó.

Để tạo tham chiếu chéo tới các đề mục (heading), sinh viên vào tab “References”, rồi tìm và nhấn chọn “Cross-refrence”. SV chọn “Refrence type” là “Heading” và chọn “Insert Reference to” là “Heading number (no context)”. Sau đó, SV chọn phần đề mục muốn tham chiếu rồi bấm “Insert”.

Để tạo tham chiếu chéo tới các hình vẽ, bảng biểu và công thức, sinh viên cũng vào tab “References” và chọn “Cross-refrence”. SV chọn “Refrence type” là “Hình”, “Bảng”, hoặc “Công thức’. SV chọn “Insert Reference to” là “Only label and number”, trỏ đến phần muốn tham chiếu rồi bấm “Insert”. Nếu font chữ trong liên kết tham chiếu tạo ra được in đậm (bold), SV chuyển về dạng thường cho chuẩn tắc.

Thực hiện tương tự các bước mô tả ở trên, sinh viên có thể tạo tham chiếu chéo tới các tài liệu tham khảo. Sinh viên chọn “Refrence type” là “Numbered item”, chọn “Insert Reference to” là “Paragraph number”, trỏ đến phần tài liệu tham khảo muốn tham chiếu rồi bấm “Insert”. Ví dụ, tham chiếu chéo tới tài liệu tham khảo [2], [3], [4] đã được tạo.

Cập nhật mục lục và tham chiếu chéo

Trong suốt quá trình viết ĐATN, sinh viên sẽ tạo ra nhiều xáo trộn như thay đổi vị trí hình và bảng, thay đổi thứ tự tài liệu tham khảo, thay đổi tên đề mục, v.v. Vì vậy, để hoàn thành ĐATN, sinh viên cần cập nhật lại các thành phần mục lục, danh sách hình ảnh/bảng/công thức và tham chiếu chéo được tạo tự động trong ĐATN của mình.

Để cập nhật các thành phần này, sinh viên bấm Ctrl+A để chọn toàn báo cáo, nhấn chuột phải và chọn “Update Field”, rồi liên tục chọn mục “Entire table” khi được hỏi trong hộp thoại Popup Menu. Sau đó sinh viên search chữ “Error” trên toàn báo cáo để kiểm tra xem có lỗi đánh chỉ mục hoặc lỗi tham chiếu nào không. Đồng thời, sinh viên nên tự soát lại bằng tay toàn bộ nội dung quyển ĐATN để tránh mọi sai sót.

In quyển đồ án tốt nghiệp

Do hiện nay có nhiều phiên bản Word cho nhiều nền tảng máy tính, sinh viên nhất thiết phải xuất ĐATN ra định dạng PDF rồi mang tới cửa hàng in ấn để tránh sai sót.

Sinh viên chỉ cần in bìa giấy cứng mà không cần đóng quyển để tiết kiệm chi phí và giúp việc vận chuyển ĐATN dễ dàng hơn.

Quyển ĐATN nên được in một mặt trên các trang giấy A4.

Đặc tả use case

Nếu trong nội dung chính không đủ không gian cho các use case khác (ngoài các use case nghiệp vụ chính) thì đặc tả thêm cho các use case đó ở đây.

Đặc tả use case “Thống kê tình hình mượn sách”

Cách trình bày tương tự như phần hướng dẫn ở mục 2.3.

Đặc tả use case “Đăng ký làm thẻ mượn”

Cách trình bày tương tự như phần hướng dẫn ở mục 2.3.

Công nghệ sử dụng

Công nghệ bảo mật dữ liệu

Công nghệ blockchain

Thiết kế gói

Thiết kế gói cho kiến trúc tổng quan

Thiết kế gói cho chức năng “Trả sách”

Thiết kế lớp

1. https://www.tensorflow.org/, lần truy cập cuối: 28/06/2018 [↑](#footnote-ref-1)