TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN TỐI ƯU HÓA CƠ SỞ DỮ LIỆU**

*Người hướng dẫn*: **GV. NGUYỄN THANH KHƯƠNG**

*Người thực hiện*: **NGUYỄN TRẦN QUANG HUY – 52000668**

Lớp **: 20050201**

Khoá  **: 24**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN TỐI ƯU HÓA CƠ SỞ DỮ LIỆU**

*Người hướng dẫn*: **GV. NGUYỄN THANH KHƯƠNG**

*Người thực hiện*: **NGUYỄN TRẦN QUANG HUY – 52000668**

Lớp **: 20050201**

Khoá  **: 24**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2022**

LỜI CẢM ƠN

Em xin gửi lời cảm ơn thầy Nguyễn Thanh Khương là giảng viên của em lớp Tối ưu hóa cơ sở dữ liệu đã truyền đạt kiến thức và hướng dẫn cho em cách làm bài sao cho tốt nhất. Một lần nữa em xin chân thành cảm ơn thầy.

Mặc dù em đã cố gắng hết sức nhưng vẫn không thể tránh khỏi sai sót trong quá trình làm bài và trình bày báo cáo, em rất mong nhận được sự phản hồi và thông cảm của thầy.

*TP. Hồ Chí Minh, ngày 08 tháng 12 năm 2022*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

Huy

Nguyễn Trần Quang Huy

TÓM TẮT

Bài báo cáo này bao gồm 5 phần: (Phần 1) Tối ưu hóa câu truy vấn (query), (Phần 2) Tìm hiểu về chủ đề …, (Phần 3) SQL Excuton Plan, (Phần 4) Mô tả và hiện thực 2 thuật toán B Tree và B+ Tree, (Phần 5) Tự đánh giá.

Các hình ảnh, tài liệu, đường link sử dụng trong bài báo cáo này được tổng hợp lại ở github của em ([@github.com:nguyenhuy158](https://github.com/nguyenhuy158/SQL-Tuning)) hoặc truy cập trực tiếp vào đường dẫn sau <https://github.com/nguyenhuy158/SQL-Tuning>.

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc121085912)

[TÓM TẮT ii](#_Toc121085913)

[MỤC LỤC iii](#_Toc121085914)

[DANH MỤC HÌNH VẼ v](#_Toc121085915)

[DANH MỤC BẢNG BIỂU vi](#_Toc121085916)

[DANH MỤC CÁC CHỮ VIẾT TẮT vii](#_Toc121085917)

[Phần 1: Thực hiện tối ưu hóa câu truy vấn 1](#_Toc121085918)

[1.1 Tình huống 1 1](#_Toc121085919)

[1.1.a Biểu diễn dưới dạng đại số quan hệ 1](#_Toc121085920)

[1.1.b Tối ưu hóa cây 2](#_Toc121085921)

[1.1.c Viết lại câu truy vấn 4](#_Toc121085922)

[1.2 Tình huống 2 5](#_Toc121085923)

[1.2.a Biểu diễn dưới dạng đại số quan hệ 6](#_Toc121085924)

[1.2.b Tối ưu cây 7](#_Toc121085925)

[1.2.c Viết lại câu tối ưu 11](#_Toc121085926)

[Phần 2: Tìm hiểu chủ đề Using Dynamic Management Objects 13](#_Toc121085927)

[2.1 Compatibility Views, Catalog Views, and DMOs 13](#_Toc121085928)

[2.2 DMO Security and Permissions 13](#_Toc121085929)

[2.3 Performance Tuning with DMOs 13](#_Toc121085930)

[2.3.a Navigating through the DMOs 13](#_Toc121085931)

[2.3.b Point-in-time versus cumulative data 13](#_Toc121085932)

[2.3.c Beware of the watcher effect 13](#_Toc121085933)

[2.3.d Using DMOs with other performance tools 13](#_Toc121085934)

[2.4 Tóm tắt 13](#_Toc121085935)

[Phần 3: SQL Execution Plan 14](#_Toc121085936)

[3.1 Tìm hiểu 14](#_Toc121085937)

[3.2 Cho ví dụ 14](#_Toc121085938)

[3.3 Giải thích 14](#_Toc121085939)

[Phần 4: Mô tả thuật toán và sử dụng Python để mô phỏng 15](#_Toc121085940)

[4.1 Cây B Tree 15](#_Toc121085941)

[4.1.a Mô tả thuật toán 15](#_Toc121085942)

[4.1.b Mô phỏng 15](#_Toc121085943)

[4.2 Cây B+ Tree 15](#_Toc121085944)

[4.2.a Mô tả thuật toán 15](#_Toc121085945)

[4.2.b Mô phỏng 15](#_Toc121085946)

[TÀI LIỆU THAM KHẢO 16](#_Toc121085947)

DANH MỤC HÌNH VẼ

[Hình 1 Câu Query chưa tối ưu hóa của tình huống 1 1](#_Toc121425337)

[Hình 2 Cây đại số quan hệ chưa tối ưu của tình huống 1 2](#_Toc121425338)

[Hình 3 2](#_Toc121425339)

[Hình 4 3](#_Toc121425340)

[Hình 5 3](#_Toc121425341)

[Hình 6 4](#_Toc121425342)

[Hình 7 5](#_Toc121425343)

[Hình 8 6](#_Toc121425344)

[Hình 9 7](#_Toc121425345)

[Hình 10 8](#_Toc121425346)

[Hình 11 8](#_Toc121425347)

[Hình 12 9](#_Toc121425348)

[Hình 13 9](#_Toc121425349)

[Hình 14 10](#_Toc121425350)

[Hình 15 10](#_Toc121425351)

[Hình 16 11](#_Toc121425352)

[Hình 17 12](#_Toc121425353)

[Hình 18 15](#_Toc121425354)

[Hình 19 Bảng sys.sysprocesses thì được tách ra thành 3 bảng như trên 15](#_Toc121425355)

[Hình 20 16](#_Toc121425356)

[Hình 21 17](#_Toc121425357)

[Hình 22 18](#_Toc121425358)

[Hình 23 18](#_Toc121425359)

[Hình 24 20](#_Toc121425360)

[Hình 25 21](#_Toc121425361)

[Hình 26 24](#_Toc121425362)

[Hình 27 26](#_Toc121425363)

[Hình 28 27](#_Toc121425364)

[Hình 29 27](#_Toc121425365)

[Hình 30 28](#_Toc121425366)

[Hình 31 28](#_Toc121425367)

[Hình 32 29](#_Toc121425368)

[Hình 33 29](#_Toc121425369)

[Hình 34 30](#_Toc121425370)

[Hình 35 30](#_Toc121425371)

[Hình 36 30](#_Toc121425372)

[Hình 37 31](#_Toc121425373)

[Hình 38 33](#_Toc121425374)

[Hình 39 33](#_Toc121425375)

[Hình 40 33](#_Toc121425376)

[Hình 41 34](#_Toc121425377)

[Hình 42 34](#_Toc121425378)

[Hình 43 37](#_Toc121425379)

[Hình 44 Thang do Bloom 6 cấp độ 39](#_Toc121425380)

DANH MỤC BẢNG BIỂU

[Bảng 1 Bảng tự đánh giá theo thang đo Bloom 39](#_Toc121425317)

DANH MỤC CÁC CHỮ VIẾT TẮT

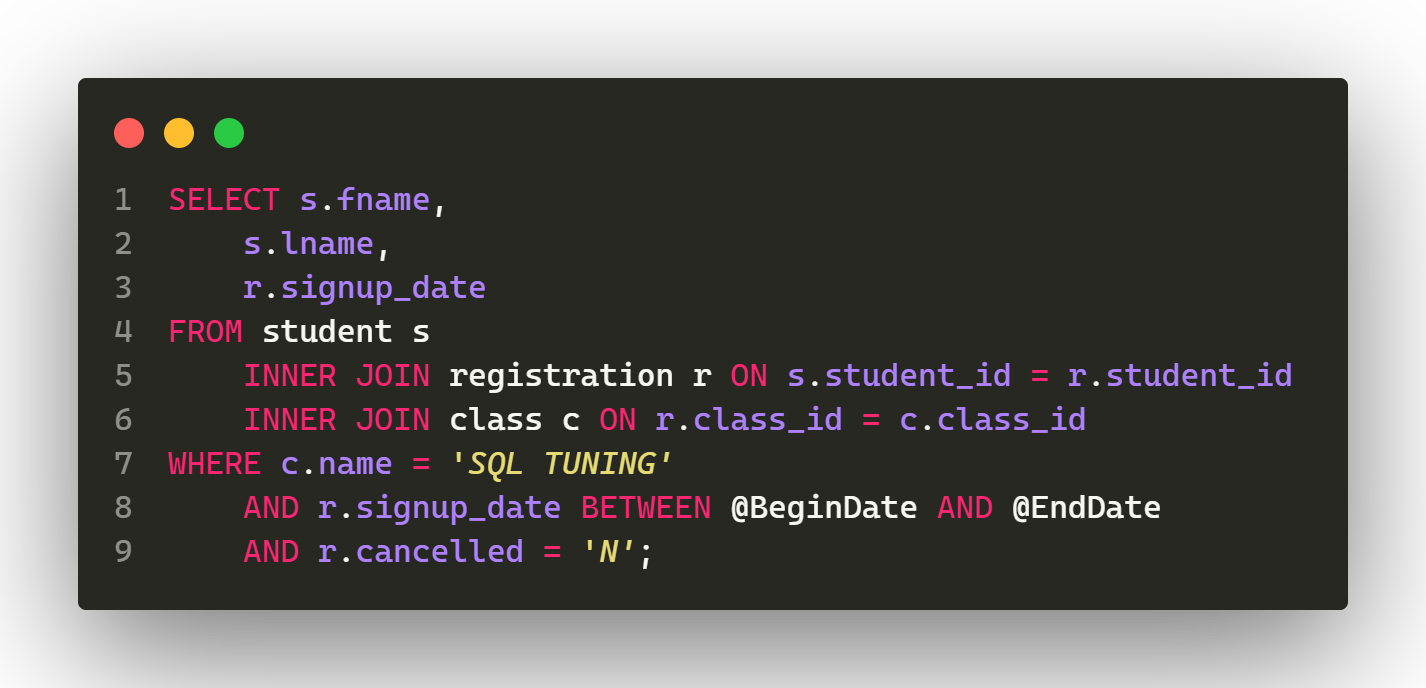
|  |  |
| --- | --- |
| SQL | Structured Query Language |
| CSDL | Cơ sở dữ liệu (Database) |
| DMOs | Dynamic Management Objects |
| DMVs | Dynamic Management Views |
| DMFs | Dynamic Management Functions |

1. Thực hiện tối ưu hóa câu truy vấn

Các bước khi tối ưu hóa một câu truy vấn, ta sẽ tối ưu hóa trên từng bảng của câu truy vấn với mỗi bảng ta sẽ đi qua lần lược các bước:

* Ngang (Phép chọn Π – WHERE)
* Dọc (Phép chiếu σ – SELECT)
* Quan hệ
  1. Tình huống 1

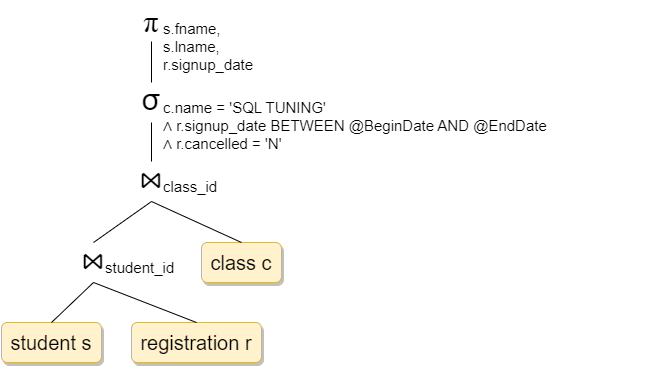
Nhân viên phòng Đào tạo thực hiện việc truy vấn CSDL để tìm ra người đăng kí môn học **“SQL TUNING”** vào **ngày cuối cùng** của đợt đăng ký với câu lệnh SQL như sau:



Hình Câu Query chưa tối ưu hóa của tình huống 1

* + 1. Biểu diễn dưới dạng đại số quan hệ

**Π** s.fname,s.lname,r.signup\_date (**σ** c.name = 'SQL TUNING' **˄** r.signup\_date BETWEEN @BeginDate AND @EndDate **˄** r.cancelled = 'N' (student s**⋈**registration r**⋈**class c))



Hình Cây đại số quan hệ chưa tối ưu của tình huống 1

* + 1. Tối ưu hóa cây

Theo cách tối ưu đã đề cập ở trên, ta có tổng cộng 3 bảng:

* class c
  + Ngang: c.name
  + Dọc:
  + Quan hệ: c.class\_id

Câu query là:

Text

Description automatically generated

Hình

* student s
  + Ngang:
  + Dọc: s.fname,s.lname
  + Quan hệ: s.student\_id

Câu query là:

Text

Description automatically generated

Hình

* registration r
  + Ngang: r.signup\_date BETWEEN @BeginDate AND @EndDate, r.cancelled = 'N'
  + Dọc: r.signup\_date
  + Quan hệ: r.class\_id, r.student\_id

Câu query là:

Text

Description automatically generated

Hình

Cây query sau khi tối ưu sẽ là:

Graphical user interface, application

Description automatically generated

Hình

* + 1. Viết lại câu truy vấn

Từ những câu query ở 1.1.b ta dùng join kết hợp lại chúng lại ta có câu query đã được tối ưu như sau:

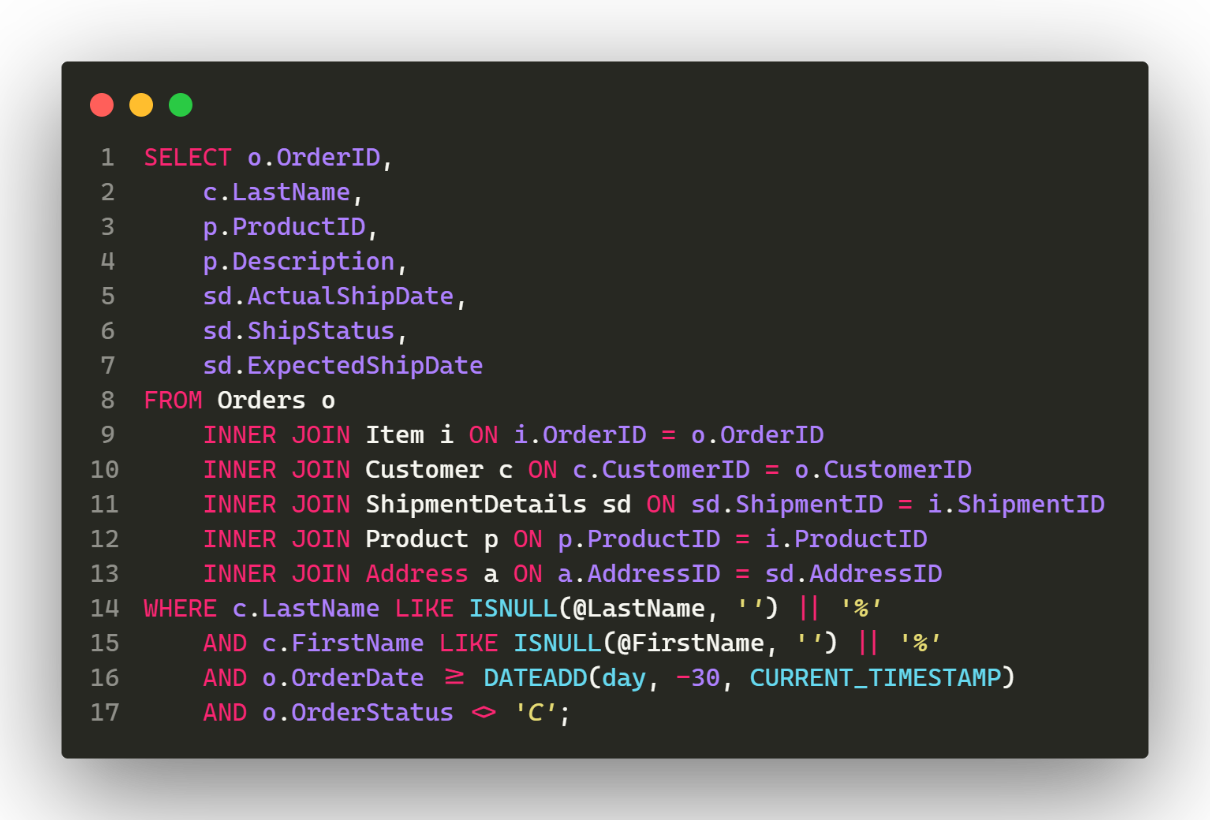
Text

Description automatically generated

Hình

* 1. Tình huống 2

Một nhân viên quản trị **CSDL** của một cửa hàng thực hiện một câu truy vấn để **liệt kê danh sách** các **đơn đặt hàng** được **mở** của khách hàng với câu lệnh SQL như sau:



Hình

* + 1. Biểu diễn dưới dạng đại số quan hệ

Cách 1:

**Π** o.OrderID, c.LastName, p.ProductID, p.Description, sd.ActualShipDate, sd.ShipStatus, sd.ExpectedShipDate(**σ** c.LastName LIKE ISNULL(@LastName, '') || '%' ˄ c.FirstName LIKE ISNULL(@FirstName, '') || '%' ˄ o.OrderDate >= DATEADD(day, -30, CURRENT\_TIMESTAMP) ˄ o.OrderStatus <> 'C' (Orders o **⋈** Item i **⋈** Customer c **⋈** ShipmentDetails sd **⋈** Product p **⋈** Address))

Cách 2:

Gọi:

* o.OrderID, c.LastName, p.ProductID, p.Description, sd.ActualShipDate, sd.ShipStatus, sd.ExpectedShipDate ***(1)***
* c.LastName LIKE ISNULL(@LastName, '') || '%'

˄ c.FirstName LIKE ISNULL(@FirstName, '') || '%'

˄ o.OrderDate >= DATEADD(day, -30, CURRENT\_TIMESTAMP)

˄ o.OrderStatus <> 'C' ***(2)***

Ta có:

**Π** (1) (**σ** (2) (Orders o **⋈** Item i **⋈** Customer c **⋈** ShipmentDetails sd **⋈** Product p **⋈** Address))

Graphical user interface

Description automatically generated

Hình

* + 1. Tối ưu cây

Theo cách tối ưu đã đề cập ở trên, ta có tổng cộng 6 bảng:

* Orders o
  + Ngang: o.OrderDate, o.OrderStatus
  + Dọc: o.OrderID
  + Quan hệ: o.OrderID, o.CustomerID

Câu query là:

Text

Description automatically generated

Hình

* Item i
  + Ngang:
  + Dọc:
  + Quan hệ: i.OrderID, i.ShipmentID, i.ProductID

Câu query là:

Text

Description automatically generated

Hình

* Customer c
  + Ngang: c.LastName, c.FirstName
  + Dọc: c.LastName
  + Quan hệ: c.CustomerID

Câu query là:

Text

Description automatically generated

Hình

* ShipmentDetails sd
  + Ngang:
  + Dọc: sd.ActualShipDate, sd.ShipStatus, sd.ExpectedShipDate
  + Quan hệ: sd.ShipmentID, sd.AddressID

Câu query là:

Text

Description automatically generated

Hình

* Product p
  + Ngang:
  + Dọc: p.ProductID, p.Description
  + Quan hệ: p.ProductID

Câu query là:

Text

Description automatically generated

Hình

* Address a
  + Ngang:
  + Dọc:
  + Quan hệ: a.AddressID

Câu query là:

Text

Description automatically generated with low confidence

Hình

Graphical user interface, application

Description automatically generated

Hình

* + 1. Viết lại câu tối ưu

Text

Description automatically generated

Hình

1. Tìm hiểu chủ đề Using Dynamic Management Objects

Chủ đề em tìm hiểu là Sử dụng đối tượng quản lý động (Using Dynamic Management Objects) trong tối ưu trên SQL Server[[1]](#footnote-1).

DMOs là một bộ object thuộc SQL Server objects, tham chiếu với lược đồ hệ thống (sys), nó cung cấp một cái nhìn về những gì đang diễn ra bên trong và các tài nguyên đang được sử dụng như thế nào. Hiểu theo một cách khác, DMOs sẽ phơi bày ra các giá trị liên quan đến connections, sessions, transactions, sql statements, và các quá trình đang và đã diễn ra, ….

DMOs được chia ra thành 2 loại là DMVs và DMFs

* DMVs là system views
* DMFs là system functions

Bên cạnh đó chúng ta có thể dùng chúng như những view và function khác có trong SQL Server (ví dụ như query, join, truyền parameter khi cần, …) để mục đích là kiểm tra những vấn đề (issue) hoặc tình trạng (health) của cơ sở dữ liệu của chúng ta hiện tại (SQL Server instance).

* 1. Compatibility Views, Catalog Views, and DMOs

*SQL Server cung cấp một lượng lớn các system views nhưng đáng để chú ý đến thì ta có 2 dạng (system compatibility views và system catalog views)*

* Chế độ xem tương thích – Compatibility Views
* Chế độ xem danh mục – Catalog Views, and DMOs

Tuy nhiên trong hầu hết các trường hợp thì chế độ xem danh mục (Catalog Views và DMOs) đã cho thấy được sử hiệu quả, tiện lợi khi sử dụng do đó gián tiếp làm cho chế độ xem tương thích (Compability Views) trở nên lỗi thời. Tại sao chúng ta lại chuyển đổi từ chế độ xem dạng tương thích (Compability Views) sang chế độ xem danh mục (Catalog Views và DMOs) là bởi vì nó (Catalog Views) cung cấp rất nhiều thông tin thú vị về metadata, cách chúng được hiển thị và trình bày, … giúp chúng ta nắm bắt thông tin dễ dàng hơn.

**Tại sao chế độ xem tương thích (Compatibility Views) chỉ tối ưu trên các phiên bản từ SQL Server 2000 (hoặc 2005) trở về trước còn với các phiên bản mới cho tới hiện nay thì ta lại được khuyên hãy nên sử dụng chế độ xem danh mục (Catalog Views) ?.**

Quay lại nhanh về lịch sử thì trước những năm 2005 thì SQL Server các metadata được lưu ở master database. Và Microsoft đã cảnh báo các lập trình viên (DBAs) là đừng chạy bất kỳ đoạn code nào ảnh hưởng đến chúng. Nhưng sự thật là càng chống thì các lập trình viên càng hứng thú và càng thâm nhập sâu hơn. Các thông tin trong bảng này đã cấp cho các họ (DBAs) có thể truy cập vào spid (mã định danh duy nhất cho phiên người dùng hiện tại), theo dõi mức sử dụng CPU, bộ nhớ (Memory), các thông tin về I/O, thông tin kết nối (như là tên máy chủ (host), tên chương trình, thông tin đăng nhập (username, password, …) và id tiến trình (process id)) và còn vô số những cột metadata quan trọng khác giúp họ (DBAs) xác định được rõ ràng những gì đang diễn ra trên phiên SQL hiện tại vào bất kỳ thời điểm nào.

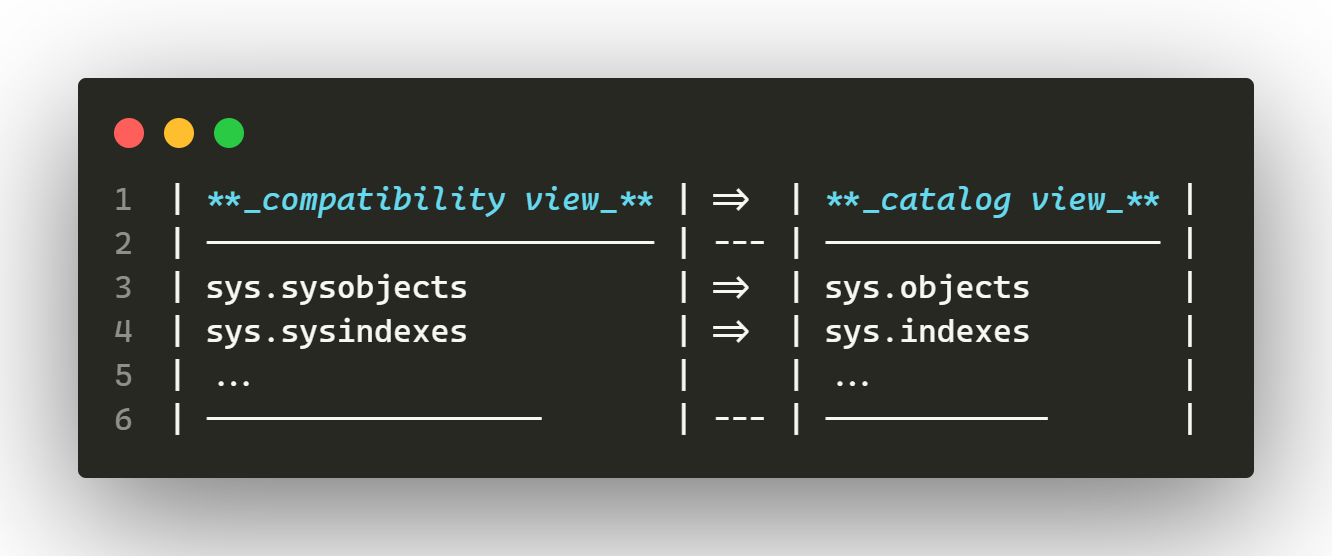
Cuối cùng Microsoft đã nhận ra sự việc quan trọng này, do đó họ đã thiết kế và giới thiệu một lớp trừu tượng mới giữa user và metadata của họ. Do đó từ 2005 tới hiện tại họ đã chuyển bảng hệ thống (system database) từ master database sang cơ sở dữ liệu mới mà ở đó các end user không thể truy cập vào được kể cả các lập trình viên (DBAs). (thật ra thì vẫn có cách nhưng trong phạm vi phần trình bày này thì sẽ không đề cập đến vấn đề đó)

Microsft đã làm như nào họ đã tạo ra một (Compatibility Views) dựa trên data của system tables và sau đó ẩn chúng đi (system tables được ẩn đi). Compatibility Views này được cấu trúc giống hệt với bảng hệ thống (system tables) cái mà đã được ẩn đi, nhưng điều khác biệt nhất lúc bấy giờ là chúng lại thuộc quyền sở hữu của lược đồ hệ thống (sys schema). Sở dĩ phải làm vậy là để phục vụ mục tiêu tương thích ngược với các phiên bản trước và do đó chúng không chứa bất kì các metadata nào mới được thêm từ phiên bản 2005 trở lên. Tuy nhiên với một lược đồ đơn giản như vậy chúng cũng giúp cho các lập trình viên (DBAs) thực hiện các câu query để khắc phục sự cố của họ.

Ngay vào lúc đó năm 2005 họ đã giới thiệu Catalog Views và DMOs đều thuộc master database và dữ liệu được lấy từ dữ liệu thực tế từ cơ sở dữ liệu. Chúng được thiết kế để hiện thị đầy đủ các chức năng mới và được khuyến khích cho các lập trình viên (DBAs) chuyển sang dùng.

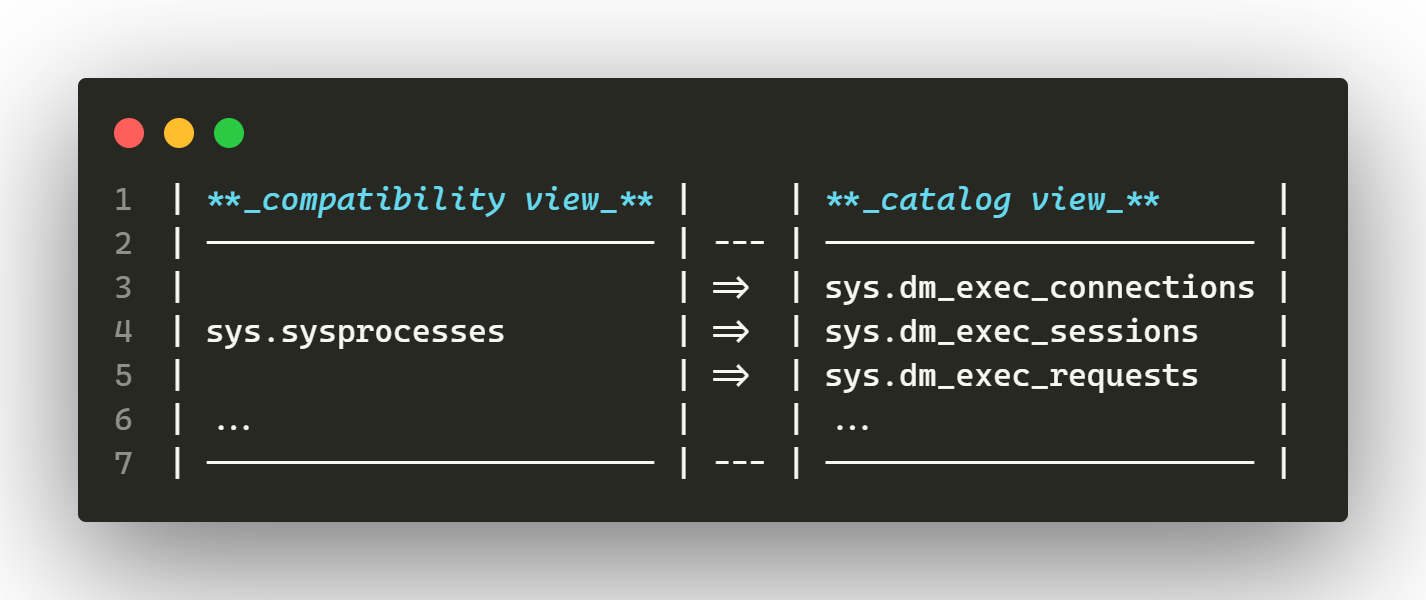
Trong quá trình chuyển đổi thì phát sinh một số vấn đề như sau:

* Một số hàm thì khi chuyển khá đơn giản (vì chúng tương đồng như nhau):



Hình

* Tuy nhiên một số hàm thì lại khá phức tạm:



Hình Bảng sys.sysprocesses thì được tách ra thành 3 bảng như trên

Ghi chú: Tuy việc tách ra làm cho mọi thứ phức tạp hóa lên tuy nhiên do sys.sysprocesses khá là phức tạp và cho nên Microsoft đã quyết định chuẩn hóa nó là giờ thì nó đã được tách ra thành 3 bảng như hình minh họa phía trên.

Và thông qua sự kiện trên Microsoft đơn giản hóa các quy ước đặt tên và chuẩn hóa lại chúng trên Catalog Views và DMOs, ví dụ một số tên đã được đổi như sau:



Hình

Việc đổi tên này được các lập trình viên (DBAs) đánh giá cao vì tính nhất quán (đọc vào là hiểu liền, dễ dàng nhìn thấy được thông tin muốn lấy ở đâu ngay lập tức).

Một điều dễ nhận thấy khi chạy các query lấy dữ liệu từ Catalog Views và DMOs là chúng ta nhận về được rất nhiều thông tin (vì đây là các metadata mới được thêm vào từ các phiên bản mới). Cùng với sự phát triển khi các công cụ ngày càng được phát triển tốt hơn hỗ trợ nhiều thứ hơn thì kéo theo đó lượng thông tin được thêm vào cũng ngành càng to ra, phìn ra.

Vấn đề tiếp theo là do ở các chế độ này thì dữ liệu được gom nhóm lại, sắp xếp theo đối tượng: ví dụ thay vì hồi trước chỉ cần sys.sysprocesses là có thể lấy được đủ thông tin như hiện tại sys.sysprocesses đã được chia thành 3 bảng cho nên lập trình viên cần xác định cụ thể là họ đang cần cái gì, giả sử họ cần thông tin các sessions đang hoạt động (sử dụng sys.dm \_ exec \_ sessions), họ cần thông tin về các requests đang chạy (sử dụng sys.dm \_ exec \_ requests), ....

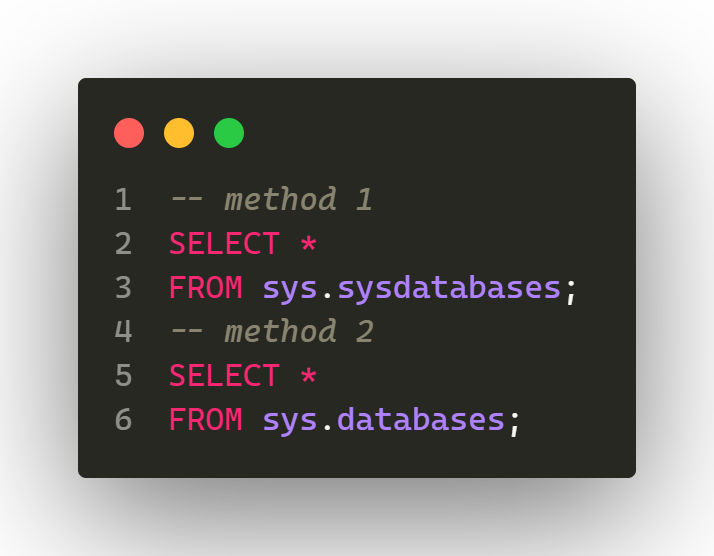
***Vậy làm thế nào để phân biệt hai loại chế độ xem tương thích (compatibility views) và chế độ xem danh mục (catalog views) ?***

Theo như chúng ta đã biết chế độ xem tương thích (Compatibility Views) và chế độ xem danh mục (Catalog Views) đều thuộc về lược đồ hệ thống (sys schema ), tuy nhiên chế độ xem tương thích (Compatibility Views) được cho phép có thể được truy cập thông qua lược đồ dbo (dbo schema), lý do để có thể tương thích ngược được với các phiên bản trước.

Ví dụ chứng minh:

Giả sử, chúng ta muốn lấy danh sách các databases (mục đích có thể là chỉ để các nhà phát triển (dev) kiểm tra thử databases đó có chưa để drop hoặc làm bước tiếp theo là gì đó, ….

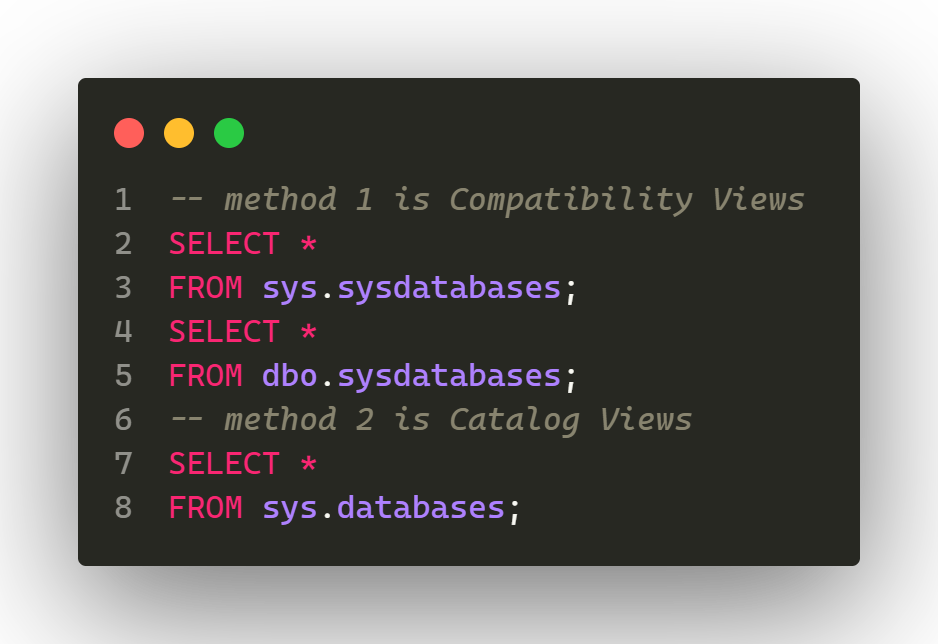
Lúc này chúng ta có 2 câu lệnh có thể sử dụng lúc này:



Hình

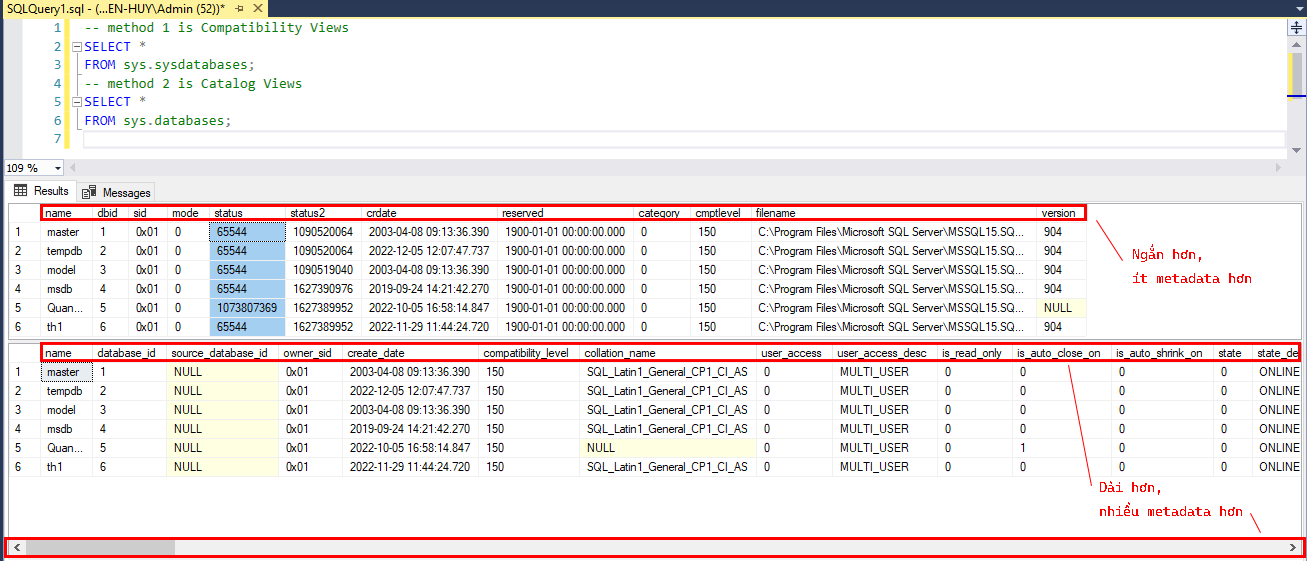
**Câu hỏi đặc ra lúc này là nếu không đi tra cứu trong sách hoặc là google có cách nào phân biệt là cái nào thuộc về chế độ xem tương thích (Compatibility Views) cái nào là chế độ xem danh mục (Catalog Views) hay không?**

Đáp án: *là thay đổi lược đồ (schema) nếu cái nào chỉ thuộc lược đồ hệ thống thôi thì chính là chế độ xem danh mục (Catalog Views) và cái nào vừa thuộc lược đồ hệ thống và lược đồ dbo thì đó chính là chế độ xem tương thích (Compatibility Views).*



Hình

Ghi chú: Một lời khuyên từ Microsoft là chúng ta nên sử dụng chế độ xem danh mục (Catalog Views) hơn thay cho chế độ xem tương thích (Compatibility Views), lý do là vì nó chứa nhiều *metadata* hơn cái còn lại (các *metadata* này chỉ được thêm vào từ các phiên bản SQL Server 2005 cho tới hiện nay).



Hình

* 1. DMO Security and Permissions

Như đã được đề thì từ phiên bản SQL 2005 trở về sau, master database không còn là cơ sở dữ liệu hệ thống chính (main system database) để lưu trữ tất cả metadata và dữ liệu hệ điều hành (operational system data).

Tất cả metadata của hệ thống (system metadata) được lưu trữ trong cơ sở dữ liệu msdb (msdb database) lấy nguồn từ cơ sở dữ liệu tài nguyên (resource database).

Cơ sở dữ liệu tài nguyên (resource database) được bảo vệ hoàn toàn kể cả người được cấp quyền system administrator, được lưu lại dưới dạng tệp mssqlsystemresource.mdf và mssqlsystemresource.ldf được lưu trong thư mục mặc định của SQL Server 2005+ (default data folder).

Không thể truy vấn hoặc sao lưu cơ sở dữ liệu tài nguyên (resource database) đó là lý do nó được bảo vệ hoàn toàn.

Master database đã chuyển đổi thành một lớp trừu tượng (abstraction layer) có thể hiểu nó là một căn nhà mà ở đó có chứa compatibility views, catalog views, and DMOs và dữ liệu cuối cùng được đưa đến end user được kiểm soát và an toàn và để có thể được quyền xem các thông tin này thì nếu bạn không phải là quản trị viên hệ thống (system administrator of the server) thì bạn phải bạn phải được cấp 1 trong 2 quyền như phía dưới (tùy thuộc vào phạm vi bạn đang truy vấn):

* VIEW SERVER STATE: dùng để truy cập các đối tượng trong phạm vi máy chủ (server scoped objects)
* VIEW DATABASE STATE: dùng để truy cập đến các đối tượng trong phạm vi cơ sở dữ liệu (database scoped objects)

Ghi chú: Ta có thể cấp quyền cho tất cả người dùng có thể truy cập DMOs mà không sợ họ xóa cơ sở dữ liệu (drop database) của chúng ta bởi vì đơn giản là họ chỉ có quyền xem mà thôi.

* 1. Cải thiện hiệu suất với (Performance Tuning with DMOs)

Khi ta có một câu truy vấn chậm thì đa số mọi người thường hay nghĩ đến việc là cần nhiều CPU, RAM, ổ cứng phải chạy nhanh hơn,... mà không thật sự biết nguyên nhân gây ra vấn đề trên đến từ đâu. (ví dụ: nếu tổng thời gian phải hồi của SQL Server của bạn là 100% thời gian trong khi đó đã tốn hết 90% cho viện I/O chỉ có 10% là dành cho CPU thì việc nâng cấp CPU lúc này rất vô nghĩa không đẹm lại mục tiêu thật sự).

Khi sử dụng DMOs giúp cho chúng ta trong việc thu hẹp phạm vi và khắc phục sự cố nhanh hơn.

Một ví dụ điển hình là:

Với mỗi lần một worker đợi một tài nguyên thì nó sẽ được SQL Server ghi nhận lại và ta có thể truy cập vào đó để xem xét ra đưa ra hướng giải quyết. (Dữ liệu này được lưu lại cho tới khi phiên bản SQL Server hiện tại khởi động lại).

Text

Description automatically generated

Hình

Graphical user interface, text, application

Description automatically generated

Hình

* + 1. Điều hướng thông qua các DMOs (Navigating through the DMOs)

Quá trình chuẩn đoán là một quá trình bóc tách các lớp mà ở đó lớp đầu tiên chúng ta gặp đến là lớp bên ngoài (bao gồm xem là ai đang kết nối, kết nối bằng cách nào, các phiên đang chạy và các yêu cầu được các phiên này thực hiện là gì) sử dụng *yêu cầu, phiên và kết nối (Connections, Sessions and Requests)* và *kế hoạch truy vấn (Query Plan Metadata).* Tiếp đến *các giao thức (Transactions).* Một khía cạnh tiếp theo không kém phần quan trọng trong quá trình tối ưu hóa là giảm thiểu số lượng I/O ta cần dùng đến c*hiến lược lập chỉ mục và bảo trì (Indexing Strategy and Maintenance)* và *thống kê và sử dụng đĩa vật lý (Physical Disk Statistics and Utilization).* Cuối cùng lớp mà các yêu cầu của chúng ta được hệ điều hành thực hiện như thế nào *hệ điều hành và tương tác phần cứng (OS and Hardware Interaction).*

Các lập trình viên (DBAs) thường sử dụng kết quả từ nhiều lớp kết hợp chúng lại và đưa ra kết quả, kết luận cuối cùng.

* + 1. Dữ liệu ngay hiện tại so với dữ liệu tích lũy (Point-in-time versus cumulative data)

Hãy luôn nhớ là bạn có thể query tới DMOs bình thường như query tới bất kỳ bảng nào. Tuy nhiên, hãy nhớ là dữ liệu được lưu trong DMOs là dữ liệu động do đó dữ liệu mà bạn thấy khi query thực chất là một ảnh chụp nhanh (snapshot) vào thời điểm đó mà thôi.

Đôi khi bạn được thông tin là hệ thống đang có sự cố tuy nhiên bạn query tới DMOs thì thông tin bạn nhận được rất bình thường không có gì là có sự cố cả thì hẫy nhớ là thông tin bạn nhận được là vào lúc đó có thể là lúc đó hệ thống chưa có bị sự cố hoặc là sự cố đã qua và chưa tới lại. Cách khắc phục là bạn có thể query nhiều lần để có thể nhìn được cụ thể tổng quát tình trạng của hệ thống hiện tại là như thế nào.

* + 1. Cẩn thận với hiệu ứng người theo dõi (Beware of the watcher effect)

Mặc dù việc query tới DMOs sử dụng rất ít tài nguyên của hệ thống, tuy nhiên cũng cần phải cẩn trọng tránh hiệu ứng người theo dõi (watcher), hiệu ứng này sẽ làm ảnh hưởng tới hiệu suất của hệ thống, khi mà câu query tới DMOs của bạn phức tạp (thu thập, tổng hợp thông tin, tìm kiếm, …) có thể bạn sẽ bị hiệu ứng người theo dõi và dẫn đến hệ thống bạn sẽ giảm hiệu suất xử lý.

* + 1. Sử dụng DMO với các công cụ hiệu suất khác (Using DMOs with other performance tools)

Tuy có những ưu điểm như vậy tuy nhiên DMO không phải là sự thay thế cho các công cụ khác như PerfMon, Profiler, …

Ví dụ: **Profiler** là một công cụ vô mạnh mẻ để theo dõi một tập hợp các hành động đang xảy ra trong SQL Server, là một công cụ mạnh mẽ để chẩn đoán các query hoạt động kém. Do đó, các lập trình viên (DBAs) nên tiếp tục sử dụng Profiler thường xuyên, cùng với các DMOs. Mặc dù DMOs cung cấp rất nhiều thông tin về trạng thái của phiên hiện tại, yêu cầu giao dịch, … về các hoạt động trong phiên hiện tại. Tuy nhiên, Profiler rất mạnh về khả năng cung cấp thông tin chi tiết theo thời gian thực về hoạt động trên phiên. Dù vậy nó vẫn có điểm yếu đó là nó không phải là một công cụ nhẹ, bạn cần phải xem xét cẩn thận nếu không thì có thể bị hiệu ứng người theo dõi (watcher) đã được đề cập.

Giám sát hiệu suất (**Performance Monitor**) cho phép sử dụng đồ họa để hiện thị một số lượng lớn chỉ số, nhưng chủ yếu ở cấp độ máy chủ, là một công cụ có giá trị và các lập trình viên (DBAs) thường sử dụng bằng chứng thu thập được từ các bộ đếm khác nhau với bằng chứng thu được từ các DMO.

**Activity Monitor** là một công cụ được tích hợp sẵn trong SSMS, sử dụng DMO (cũng như bộ đếm PerfMon). Sử dụng giao hiện đồ họa biểu diễn % thời gian CPU, số lượng tác vụ đang chờ, I/O MB/giây, …. Là một nơi không tồi để bắt đầu kiểm tra hiệu suất.

Cuối cùng, các công cụ giám sát của bên thứ 3 có bảng điều khiển sẽ liên kết trực tiếp với DMOs. Những công cụ này có thể loại giúp giảm bớt một số khó khăn khi trong việc khắc phục sự cố. Họ sẽ lấy dữ liệu ra từ DMOs sau đó trình bày dữ liệu đó ở dạng trực quan hơn nhiêu và quan trọng là chúng có thể cảnh báo trước các sự cố sắp xảy ra. Đánh đổi lại bạn mất một công cụ chắc chắn đó là khả năng kiểm soát sự chính xác của dữ liệu được trả về mà bạn nhận được điều này chỉ có khi chính bạn viết các truy vấn cho riêng mình.

1. SQL Execution Plan
   1. Tìm hiểu

Để có thể thực hiện được một câu query, SQL Server Database cần phải phân tích các câu lệnh để quyết định cách hiệu quả nhất để truy cập vào các dữ liệu cần thiết. Quá trình phân tích này được thực hiện bởi công cụ là trình tối ưu hóa câu query (Query Optimizer). Đầu vào của query optimizer chứa những thông tin như sau query, lược đồ database (the database schema), …. Kết quả của quá trnh này được gọi là **query execution plan**, đôi khi liên quan đến query plan.

**Query execution plan** là:

* *Trình tự truy cập vào các bảng:*

Cụ thể là có rất nhiều cách để database server truy cập vào các bảng (tables) để tạo ra kết quả. Ví dụ nếu ta chạy một câu query cần dữ liệu từ 3 bảng TableOne, TableTwo, TableThree thì cách trường hợp có thể xảy ra là:



Hình

Có thể truy cập vào TableOne sau đó nối với TableTwo rồi sau cùng nối tới TableThree, hoặc là những cách còn lại.

* *Những phương thức (methods) sẽ được sử dụng để lấy dữ liệu từ các bảng:*

Nhìn chung là có rất nhiều cách để lấy dữ liệu từ một bảng. Nếu chỉ lấy một vài dòng mà còn biết được cụ thể khóa của dòng đó thì database server sử dụng index. Trường hợp nếu muốn lấy hết tất cả các dòng của một bảng database server có thể sẽ không sử dụng index mà sẽ sử dụng quét bảng (scan table). Trong trường hợp các bảng của bạn quá nhỏ (ít dòng dữ liệu) thì database server thường sẽ sử dụng quét bảng (scan table) bởi vì đó là cách hiệu quả nhất trong trường hợp này.

* *Các phương thức (methods) dùng để tính toán, lọc dữ liệu (filter), sắp xếp dữ liệu.*

Việc này sẽ xảy ra khi trong câu query của bạn có các câu lệnh như là GROUP BY hoặc ORDER BY, WHERE hoặc HAVING, ...

Lưu ý: SQL Server Management Studio có tổng cộng 3 cách để hiện thị execution plans:

* **Estimated Execution Plan**: giống như tên gọi nó là một bảng kết hoạch được Query Optimizer xây dựng nên mang tính chất dự kiến.
* **Actual Execution Plan:** nó cũng là một bảng kết hoạch nhưng kèm với một execution context (khi user chạy một câu query thì sẽ tạo nên một cấu trúc thực thi của chúng, bao gồm tham số chẳng hạn thì khái niệm dùng để miêu tả cấu trúc trên đó là execution context). Dữ liệu này chỉ khả dụng khi câu query được thực thi xong. Bao gồm thông tin về thời gian chạy thực tế (actual runtime information), các cảnh báo (warning) nếu có, thời gian sử dụng CPU, ….
* **Live Query Statistics:** giống như Actual Execution Plan nhưng dữ liệu khả dụng khi câu query đang được thực thi và được cập nhật liên tục. Cũng bao gồm thông tin về thời gian chạy, số dòng đã được sử dụng, toán tử sử dụng, ước lượng về quá trình truy vấn.

Vậy tóm lại giải thích một cách dễ hiểu **Execution plan** là nó giống như dạng debug vậy nó show từng step step của câu query (cost, time, …) đi qua bảng nào bảng nào rồi với mỗi bảng làm gì, join như thế nào (nếu có), chi phí bỏ ra như nào, … Và được trình bày với dạng giao diện đồ họa rất dễ để theo dõi, quan sát và sử dụng.

SQL có cung cấp cho chúng ta một công cụ hỗ trợ khá đắc lực đó là Execution Plan. Chúng ta sẽ kích hoạt Execution Plan trước khi chạy câu query bằng phím tắt Ctrl+M trong màn hình SQL Server Management Studio.

* 1. Cho ví dụ và tối ưu

Ta có một cơ sở dữ liệu (database) quản lý của một cửa hàng như sau



Hình

Cơ sở dư liệu trên là về một cửa hàng tên là No Brand Shop một yêu cầu đặc ra lúc này của công ty là cần thông tin về tên của sản phẩm (*ProductName*), nhà sản xuất của sản phẩm đó (*CompanyName*), mã định danh đơn hàng (*OrderID*), tên của nhân viên xử lý của những đơn hàng thuộc tính tới hiện tại được hơn 25 năm tuổi, có giảm giá (*Discount*) lớn hơn 0.15 và trong tên những công ty sản xuất có các chữ cái như H, U, Y.

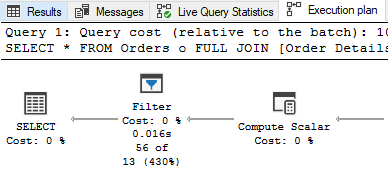
Một lập trình viên đã viết câu truy vấn như sau:

Graphical user interface, text

Description automatically generated

Hình

Sau khi chạy câu query trên trong SQL Server thì ta nhận được kết quả như sau:



Hình

Graphical user interface, application, Excel

Description automatically generated

Hình

Tốn trung bình khoảng 0.016s mặt dù con số không lớn lắm. Tuy nhiên với cơ sở dữ liệu mẫu của chúng ta chỉ 2000 dòng thì con số này không nhỏ lắm.

A picture containing diagram

Description automatically generated

Hình

Diagram, schematic

Description automatically generated

Hình

. Ta cùng nhìn lại *Actual Execution Plan* và *Live Query Statistics* ta thấy có vài vấn đề ở đây là chúng toàn sử dụng Table Scan và đọc hết các dòng có trong các bảng mà ta truy vấn từ đó ta sẽ dựa vào các thông tin đó tối ưu hóa câu truy vấn và cơ sở dữ liệu.



Hình

Sau khi đánh index cho các bảng, nối khóa ngoại ta chạy lại thử:

Graphical user interface, diagram, schematic

Description automatically generated

Hình

Graphical user interface, text, application

Description automatically generated

Hình

Kết quả là thời gian giảm được một chút 0.016 xuống 0.007. Tiếp theo ta sẽ tối ưu hóa câu query và chạy lại thử.

Graphical user interface

Description automatically generated

Hình

Graphical user interface, diagram

Description automatically generated with medium confidence

Hình



Hình

Text

Description automatically generated

Hình

Graphical user interface, text

Description automatically generated

Hình

Tới đây có vẻ đã nhanh hơn lúc ban đầu 0.016 xuống còn 0.004 giảm được 4 lần. Cách tối ưu query thì cũng làm tương tự giống phần 1.

Graphical user interface, diagram

Description automatically generated

Hình

Diagram

Description automatically generated

Hình

1. Mô tả thuật toán và sử dụng Python để mô phỏng
   1. Cây B Tree

B-tree là một dạng đặc biệt của cây tìm kiếm tự cân bằng (self-balancing search tree) mà ở đó mỗi node có thể chứa nhiều hơn một khóa và mỗi node cũng có thể chứa nhiều hơn nhiều hơn 2 node con trở lên. Có thể nói B-Tree là một dạng tổng quát hơn của cây tìm kiếm nhị phân.

*Tại sao chúng ta lại cần B Tree để làm gì?*

Cây b-tree ra đời phục vụ việc tốn ít thời gian hơn khi truy cập vào đĩa cứng.

Các cấu trúc dữ liệu khác như cây tìm kiếm nhị phân, cây avl, cây đỏ đen, v.v. chỉ có thể lưu trữ một khóa trong một nút. Nếu bạn phải lưu trữ một số lượng lớn các khóa, thì chiều cao của những cây như vậy sẽ trở nên rất lớn và thời gian truy cập sẽ tăng lên.

Tuy nhiên, B-tree có thể lưu trữ nhiều khóa trong một nút và có thể có nhiều nút con. Điều này làm giảm đáng kể chiều cao cho phép truy cập đĩa nhanh hơn.

*Các thuộc tính của một cây B Tree là gì?*

* với mỗi node x, các khóa sẽ được lưu theo thứ tự tăng dần.
* Trong mỗi node, x.leaf = true nếu x là node lá và ngược lại
* Nếu n là bậc của cây, thì mỗi node trong () có thể chứa tối đa n - 1 khóa và các con trỏ tới các node con
* Mỗi nút trừ gốc có thể có nhiều nhất n nút con và ít nhất n/2 nút con.
* Tất cả các lá có cùng độ sâu.
* Root có ít nhất 2 con và chứa tối thiểu 1 khóa.
* If n ≥ 1, then for any n-key B-tree of height h and minimum degree t ≥ 2, h ≥ logt (n+1)/2.
  + 1. Mô tả thuật toán
       1. Tìm kiếm

tìm kiếm một phần tử trong b-tree

tìm kiếm trong b-tree cũng giống như trong binary search tree bao gồm những bước như sau (gọi k là phần tử muốn tìm kiếm)

Bắt đầu từ node gốc, so sánh k với khóa đầu tiên của node .

Nếu k = khóa đầu tiên của node , trả về node và chỉ mục.

Nếu k.leaf = true, trả về NULL (nghĩa là không tìm thấy).

Nếu k < khóa đầu tiên của node gốc, tìm kiếm con trái của khóa này một cách đệ quy.

Nếu có nhiều hơn một khóa trong nút hiện tại và k > khóa đầu tiên, hãy so sánh k với khóa tiếp theo trong node .

Nếu k < phím tiếp theo, hãy tìm kiếm con bên trái của khóa này (nghĩa là k nằm giữa khóa thứ nhất và khóa thứ hai).

Khác, tìm kiếm con bên phải của khóa.

Lặp lại các bước từ 1 đến 4 cho đến khi chạm đến chiếc lá.

độ phức tạp (Searching Complexity on B Tree)

* + - 1. Chèn

chèn một phần tử trong b-tree

chèn một phần tử trong b-tree gồm 2 thao tác: tìm vị trí thích hợp và tác node nếu cần thiết

các bước chèn một phần tử

Nếu cây trống, tạo một nút gốc và chèn khóa vào vị trí đầu tiên.

Cập nhật số lượng khóa được chèn của node.

Tìm kiếm node thích hợp để chèn.

Nếu node đầy, hãy làm theo các bước bên dưới.

Chèn các phần tử theo thứ tự tăng dần.

Bây giờ, số lượng khóa đã vượt quá số lượng khóa tối đa của một node. Vì vậy, chia ở trung vị.

Đẩy khóa ở giữa lên trên và đặt các khóa bên trái làm con bên trái và các khóa bên phải làm con bên phải.

Nếu node không đầy, hãy làm theo các bước bên dưới.

Chèn node theo thứ tự tăng dần.

độ phức tạp

Ta sẽ làm một ví dụ chèn với dãy số sau:

Graphical user interface, background pattern

Description automatically generated

Hình

Ta cùng bắt đầu chèn từng số

A picture containing application

Description automatically generated

Hình Chèn 18 và 57

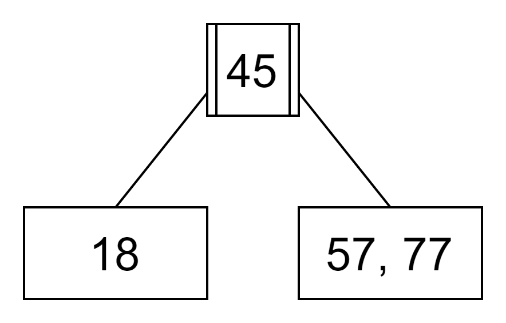
Lúc này mọi thứ vẫn bình thường chưa có gì vi phạm quy định

A picture containing icon

Description automatically generated

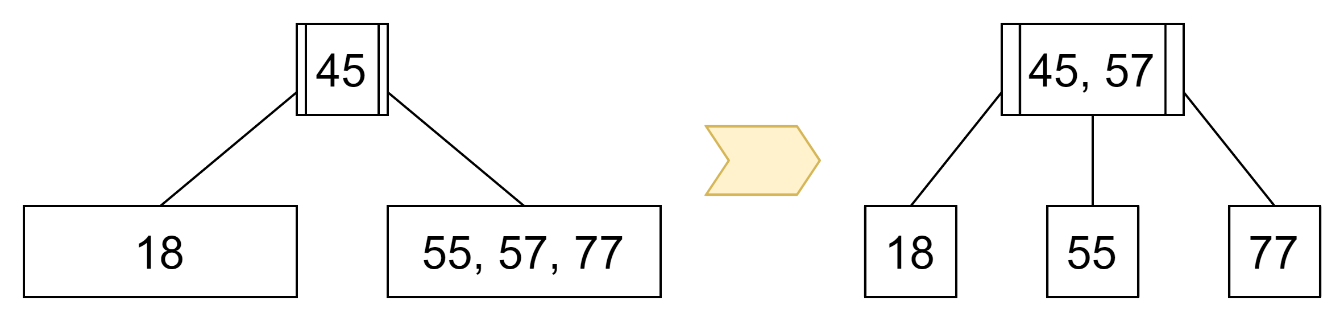
Hình Chèn 45

Lúc này đã có sự vi phạm node đầu tiên có 3 phần tử > m – 1 (3 – 1 = 2) vi phạm là mỗi node chỉ chứa được tối đa m – 1 khóa mà thôi. Xử lý ta lấy phần tử ở giữa đưa lên trên làm một node cha mới.



Hình Chèn 77

77 chèn không gây ra vi phạm gì cả



Hình Chèn 55

Node lá bên phải của node 45 đã mang 3 phần tử > 2 phần tử như đã nêu ở trên ta tiếp tục tách phần tử ở giữa đưa lên trên làm node mới nhưng phía trên đã có node rồi nên ta chèn vào thôi chứ không cần tạo node mới chi nữa.

Application, calendar

Description automatically generated

Hình Chèn 68

Mọi thứ vẫn bình thường

Graphical user interface, application

Description automatically generated

Hình Chèn 70

Lúc này node con bên phải của node gốc (45, 57) đã chứa 3 phần tử vi phạm, ta xử lý tách khóa giữa là 70 đưa lên trên. Tới đây lại node cha lại vi phạm nên ta lại tiếp tục tách phần tử ở giữa node cha lên làm node cha của node cha hiện tại. Ta có kết quả được như hình cuối cùng (gốc dưới bên phải).

Application, Teams

Description automatically generated with medium confidence

Hình Chèn 20

Chèn 20 vẫn bình thường không vi phạm.

Application, Teams

Description automatically generated with medium confidence

Hình Chèn 48

Chèn 48 vẫn bình thường không có vi phạm.

Application, Teams

Description automatically generated with medium confidence

Hình Chèn 86

Chèn 86 vẫn bình thường không có vi phạm.

A picture containing application

Description automatically generated

Hình Chèn 21

Sau khi chèn 21 thì node con bên trái của node giữa (internal node) 45 lại vi phạm chứa 3 khóa nên ta xử lý tách khóa giữa của node vi phạm đưa lên node cha. Ta có kết quả như phía tay phải.

Graphical user interface, application, Teams

Description automatically generated

Hình Chèn 43

Chèn 43 mọi thứ vẫn bình thường.

Graphical user interface, application, Teams

Description automatically generated

Hình Chèn 62

Chèn 62 mọi thứ vẫn bình thường.

A screenshot of a computer

Description automatically generated with medium confidence

Hình Chèn 87

Chèn 87 thì node con bên phải của node giữa 70 lại chứa 3 khóa vi phạm nguyên tắc, nên ta xử lý tách khóa giữa của node vi phạm đưa lên node cha.

Graphical user interface, application, Teams

Description automatically generated

Hình Chèn 69

Sau khi chèn 69 thì node con bên trái của node giữa (70, 86) vi phạm vì có 3 phần tử ta xử lý tách khóa giữa node vi phạm đưa lên node cha. Mà sau khi ta đưa lên node cha thành (68, 70, 86) số lượng khóa là 3 lại vi phạm, nên ta lại tiếp tục tách khóa giữa đẩy lên node cha tiếp. Cuối cùng ta có kết quả như trên.

* + - 1. Xóa
    1. Mô phỏng

Source code mô phỏng nằm ở thư mục ***Phần 4 (Part 4) > Demo B Tree***

File ***README.md***là file mô tả là thuật toán làm được những gì và cần làm gì để chạy thuật toán.

* 1. Cây B+ Tree

B+ Tree là một dạng nâng cao của cây tự cân bằng (self-balancing tree) mà ở đó tất cả các giá trị hay còn gọi là các khóa đều nằm ở các node lá (leaf).

Để hiểu rõ B+ Tree ta còn phải hiểu rõ một khái niệm nữa đó là chỉ mục đa cấp (multilevel indexing), trong đó các chỉ mục (index) In multilevel indexing, the index of indices is created as in figure below. It makes accessing the data easier and faster.

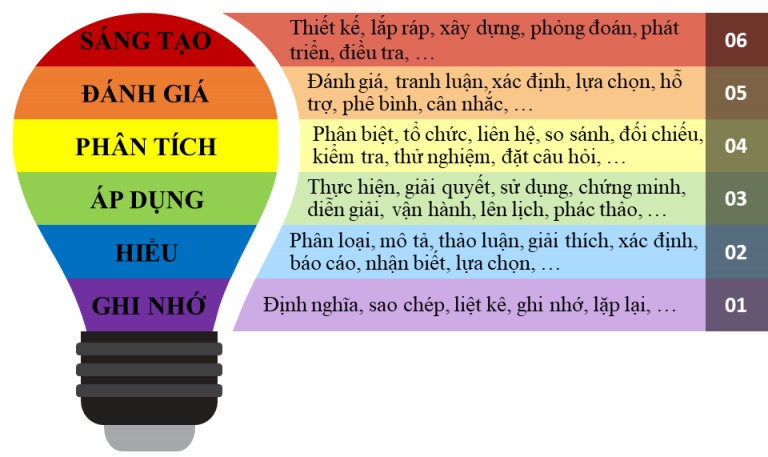
* + 1. Mô tả thuật toán
       1. Tìm kiếm
       2. Chèn
       3. Xóa
    2. Mô phỏng

Source code mô phỏng nằm ở thư mục ***Phần 4 (Part 4) > Demo B Plus Tree***

File ***README.md*** là file mô tả là thuật toán làm được những gì và cần làm gì để chạy thuật toán.

1. Tự đánh giá

Em sử dụng thang đo Bloom để đánh giá bài báo cáo của mình (*Thang đo Bloom bao gồm 6 cấp độ từ thấp đến cao như sau: 1. Ghi nhớ (Remembering), 2. Hiểu (Understanding), 3. Áp dụng (Applying), 4. Phân tích (Analyzing), 5. Đánh giá (Evaluating), 6. Sáng tạo (Creating)*)

[](https://thinkingschool.vn/wp-content/uploads/2021/03/2-0315-Lightbulb-Diagram-Six-Steps-PGo-4_3-2-768x457.png)

Hình Thang do Bloom 6 cấp độ

Bảng Bảng tự đánh giá theo thang đo Bloom

|  |  |  |
| --- | --- | --- |
| Nội dung các phần | Tự đánh giá | Cảm nhận |
| Phần 1 |  |  |
|  |  |  |
| Phần 2 |  |  |
| Phần 3 | Áp dụng và có một ít Phân tích | Về phần này em thấy mình biết ứng dụng Execution plan vào thực tế. |
| Phần 4 | B Tree: Áp dụng  B+ Tree: Hiểu và có một ít Áp dụng | Phần này đối với thì cả 2 thuật toán em hiểu cách thức hoạt động tuy nhiên với B+ Tree thì có một xíu khó khăn kiến cho quá trình hoàn thành bài không được trọn vẹn. |

TÀI LIỆU THAM KHẢO

**Tiếng Anh**

Davidson, L., & Ford, T. (2010). Performance Tuning with SQL Server Dynamic Management Views. Red gate books.

Microsoft. (n.d.). Sql Server Samples. Retrieved from Github: https://github.com/microsoft/sql-server-samples

1. Chương 1 trong (Davidson & Ford, 2010) [↑](#footnote-ref-1)