

DẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



**BÁO CÁO BÀI TẬP LỚN
HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU (CO3001)**

**Tìm hiểu, Luyện tập và Hiện thực trên
Microsoft SQL Server và RavenDB**

GV hướng dẫn: PGS.TS Võ Thị Ngọc Châu

Nhóm: 14

SV thực hiện: Nguyễn Hữu Khang 2011365

Nguyễn Thị Minh Châu 2010951

Trần Sách Nhật 2014009

Nguyễn Văn Nhật Tiến 1915486

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 12/2023

MỤC LỤC

1	Giới thiệu	6
2	Microsoft SQL Server	7
2.1	Giới thiệu tổng quan	7
2.1.1	Sơ lược Microsoft SQL Server	7
2.1.2	Kiến trúc Microsoft SQL Server	8
2.1.3	Microsoft SQL Server hiện tại	9
2.2	Indexing	11
2.2.1	Giới thiệu	11
2.2.2	Index trong Microsoft SQL Server	11
2.2.2.1	Clustered	11
2.2.2.2	Hash	14
2.2.2.3	Memory-Optimized Nonclustered	15
2.2.2.4	Non-clustered	16
2.2.2.5	Unique	18
2.2.2.6	Columnstore	18
2.2.2.7	Một số loại Index khác	19
2.3	Query processing và optimization	21
2.3.1	Query processing	21
2.3.1.1	Phân tích cú pháp(Parsing)	21
2.3.1.2	Đại số hóa(Algebrizing)	21
2.3.2	Query optimization	22
2.3.2.1	Giới thiệu	22
2.3.2.2	Chiến lược tối ưu	22
2.4	Transaction processing	27
2.4.1	Giới thiệu	27
2.4.1.1	Dịnh nghĩa	27
2.4.1.2	Transaction States	27
2.4.1.3	Phân loại Transaction trong SQL-Server	27
2.4.1.4	Đặc điểm của Transaction	28
2.4.2	Hiện thực hóa Transaction	28
2.4.2.1	Autocommit	28
2.4.2.2	Implicit Transaction	28
2.4.2.3	Explicit Transaction	30
2.5	Concurrency control	34
2.5.1	Các mức độ	34
2.5.2	Cơ chế khóa trong công cụ cơ sở dữ liệu	36
2.5.2.1	Độ chi tiết và cấu trúc phân cấp khóa	37
2.5.2.2	Chế độ khóa	38
2.5.2.3	Tương thích khóa	43



2.5.2.4	Chuyển đổi khóa	43
2.5.3	Deadlock	44
2.5.4	Mức cách ly dựa trên phiên bản hàng trong SQL Server	45
2.6	Backup và recovery	48
2.6.1	Xây dựng giải pháp High Availability và Disaster Recovery	48
2.6.1.1	Những lý do dẫn đến máy chủ bị hỏng	48
2.6.1.2	Sự ảnh hưởng nếu không có giải pháp Database Backup	48
2.6.2	Giải pháp Database Backup trong Microsoft SQL Server	48
2.6.2.1	Full Backup	48
2.6.2.2	Differential Backup	49
2.6.2.3	Transaction Log Backup	49
2.6.3	Xây dựng Backup Strategy	49
2.6.3.1	Recovery Point Objective(RPO)	49
2.6.3.2	Recovery Time Objective(RTO)	49
2.6.3.3	Backup Strategy	49
2.6.4	Thực hành Backup và Recovery	49
2.6.4.1	Backup Full Database	49
2.6.4.2	Insert một Record mới lần 1	50
2.6.4.3	Thực hiện Differential Backup Database	50
2.6.4.4	Insert một Record mới lần 2	51
2.6.4.5	Thực hiện Transaction Log Backup Database	51
2.6.4.6	Restore Database	51
3	RavenDB	54
3.1	Giới thiệu tổng quan	54
3.1.1	Nguồn gốc và đặc điểm của RavenDB	54
3.1.2	Tổ chức dữ liệu	55
3.1.2.1	Cluster và Node	55
3.1.2.2	Document Store - Document Session	56
3.1.2.3	Document	56
3.1.2.4	Collection	57
3.1.2.5	Mã định danh duy nhất	57
3.1.3	Lưu trữ dữ liệu	58
3.2	Indexing	59
3.2.1	Quá trình lập chỉ mục	59
3.2.2	Lucene Indexes	60
3.2.3	Map Indexes	61
3.2.4	Map Reduce Indexes	63
3.2.5	Auto Indexes	64
3.2.6	Static Indexes	66
3.2.7	Một số chỉ mục khác	68
3.2.8	Index state and prioritization	68
3.3	Query processing và optimization	70
3.3.1	Query optimizer - Quản lý indexing trong RavenDB	70
3.3.2	RavenDB Query Language - RQL	70
3.4	Transaction processing	76
3.4.1	Unit of work pattern	76
3.4.2	Identity map pattern	76
3.4.3	Batching & Transactions	76
3.4.3.1	Batching	76
3.4.3.2	Transactions	76
3.4.3.3	Transaction mode	77
3.4.4	Single-Node transaction	77
3.4.5	Cluster-Wide transaction	77
3.5	Concurrency control	78
3.5.1	Change vector	78
3.5.2	Các change vector được sử dụng như thế nào để xác định thứ tự?	78



3.5.3	Concurrency Control	78
3.6	Backup và Recovery	80
4	Ứng dụng sử dụng Microsoft SQL Server	81
4.1	Mô tả yêu cầu	81
4.2	Phân tích yêu cầu	82
4.2.1	Các kiểu thực thể mạnh và mối liên kết	82
4.2.2	Các kiểu thực thể yếu và mối liên kết	82
4.2.3	Các thuộc tính và mô tả	83
4.2.4	Các ràng buộc	83
4.2.4.1	Ràng buộc về khóa	83
4.2.4.2	Ràng buộc về cấu trúc	84
4.2.5	Thiết kế cơ sở dữ liệu ý niệm	86
4.2.6	Thiết kế cơ sở dữ liệu luân lý và định nghĩa các ràng buộc	87
4.2.6.1	Lược đồ cơ sở dữ liệu luân lý	87
4.2.6.2	Định nghĩa các ràng buộc	87
4.3	Hiện thực cơ sở dữ liệu (Database) trên Microsoft SQL Server	89
4.4	Kiến trúc ứng dụng	89
4.4.1	Mô tả sơ bộ về các công nghệ sử dụng	89
4.4.2	Mô tả sơ bộ về kiến trúc ứng dụng	90
4.5	Github repository	92
4.6	Thiết kế và hiện thực Database trên RavenDB	93
4.6.1	Mô hình hóa dữ liệu	93
4.6.2	Hiện thực truy vấn	94
4.6.2.1	Lệnh Insert	94
4.6.2.2	Lệnh Delete	95
4.6.2.3	Lệnh Update	95
4.6.2.4	Truy vấn với single condition	95
4.6.2.5	Truy vấn với composite condition	97
4.6.2.6	Truy vấn với join	98
4.6.2.7	Truy vấn với subquery	99
4.6.2.8	Truy vấn với aggregate function	99
5	Tổng kết	101
5.1	Dường dẫn video báo cáo và repository ứng dụng	101
5.2	Tóm tắt công việc đã làm	101
5.3	Dánh giá kết quả đạt được	101
5.3.1	Đối với Microsoft SQL Server	101
5.3.2	Đối với RavenDB	102
Tài liệu tham khảo		102

DANH SÁCH HÌNH VẼ

2.1	MSSQL Server Introduction	7
2.2	MSSQL Server Architecture	8
2.3	Cấu trúc Clustered Index	11
2.4	Biểu tượng quy trình tìm kiếm Clustered Index	12
2.5	Kết quả Clustered Index Seek	13
2.6	Kết quả truy vấn kích thước bản ghi min-max	14
2.7	Kết quả kiểm tra lại kích thước bản ghi max	14
2.8	Kiến trúc Hash	14
2.9	Delta Records	15
2.10	Split Page	16
2.11	Merge Page	16
2.12	Kiến trúc Non-clustered Index	17
2.13	4 bước xử lý truy vấn trong Microsoft SQL Server	21
2.14	Xem Missing Index	24
2.15	Xem Missing Index	25
2.16	Transaction states	27
2.17	Kết quả dùng Autocommit	28
2.18	Kết quả Implicit Transaction khi OFF	29
2.19	Kết quả Implicit Transaction khi ON nhưng chưa Commit	30
2.20	Kết quả Implicit Transaction khi ON và Commit	30
2.21	Kết quả dùng Explicit Transaction cập nhật năm xuất bản cuốn sách	31
2.22	Kết quả khi thêm COMMIT TRAN	31
2.23	Kết quả khi sử dụng ROLLBACK TRAN	32
2.24	Kết quả khi sử dụng Save Point	33
2.25	Dữ liệu bảng BOOK ban đầu	34
2.26	Lỗi đọc sai dữ liệu	35
2.27	Lỗi đọc ma	36
2.28	Các tài nguyên có thể khóa	37
2.29	Bảng tương thích khóa của các chế độ khóa thường gấp	43
2.30	Kết quả kiểm tra File Full Backup	50
2.31	Kết quả thêm một quyền sách mới lần 1	50
2.32	Kết quả kiểm tra File Differential Backup	51
2.33	Kết quả thêm một quyền sách mới lần 2	51
2.34	Kết quả kiểm tra File Transaction Log Backup	51
2.35	Kết quả kiểm tra File DBMSAssignmentOrderBook.mdf trước khi xoá	52
2.36	Kết quả kiểm tra File DBMSAssignmentOrderBook.mdf sau khi xoá	52
2.37	Database Engine không thể truy cập được vì sự cố	52
2.38	Cách vào Restore Database	53
2.39	Thực hiện chọn các File để Restore Database	53
2.40	Kết quả Restore thành công	53



2.41	Kiểm tra dữ liệu	53
3.1	Cluster và Node trong RavenDB	56
3.2	Document và Collection trong RavenDB	57
3.3	Minh họa cây tài liệu - Các giá trị nhỏ	58
3.4	Quá trình lập chỉ mục	59
3.5	Chỉ mục cũ	60
3.6	Minh họa Lucene Indexes	61
3.7	Quy trình trong Map Reduce Index	64
3.8	Kết quả truy vấn trong Studio	66
3.9	Kết quả truy vấn lần thứ 2	66
3.10	Các tùy chọn cơ bản hiển thị trong Studio	66
3.11	Các tùy chọn nâng cao trong Studio	67
4.1	Lược đồ E-ERD	86
4.2	Lược đồ cơ sở dữ liệu luận lý	87
4.3	Lược đồ kiến trúc ứng dụng: MVC	90
4.4	Trang manage books	91
4.5	Trang shop	91
4.6	Lược đồ mô phỏng cơ sở dữ liệu	93
4.7	Tổ chức và quản lý các document trong cơ sở dữ liệu DBMS_RavenDB	94
4.8	Kết quả truy vấn	96
4.9	Mô phỏng chỉ mục	96
4.10	Index Term	96
4.11	Kết quả truy vấn tương tự	97
4.12	Kết quả truy vấn	97
4.13	Mô phỏng chỉ mục	98
4.14	Index Term	98
4.15	Kết quả truy vấn	99
4.16	Kết quả truy vấn	99
4.17	Kết quả truy vấn	100

CHƯƠNG 1

GIỚI THIỆU

CHƯƠNG 2

MICROSOFT SQL SERVER

Dể thuận tiện cho người đọc, ở chương 2 này, một số nội dung sẽ được sử dụng từ chuyên ngành bằng tiếng Anh với ý nghĩa như nhau thay vì chỉ giới hạn sử dụng tiếng Việt.

2.1 Giới thiệu tổng quan

RDBMS là viết tắt của Hệ thống quản lý cơ sở dữ liệu quan hệ(Relational Database Management Systems) dựa trên mô hình quan hệ. Về cơ bản, nó là một chương trình cho phép chúng ta tạo, xóa và cập nhật cơ sở dữ liệu quan hệ. Dữ liệu trong RDBMS được cấu trúc trong các bảng, trường và bản ghi cho phép chúng ta lưu trữ và truy xuất dữ liệu ở định dạng bảng được tổ chức dưới dạng hàng và cột. Mỗi bảng trong RDBMS bao gồm các hàng của bảng cơ sở dữ liệu. Đây là cơ sở cho SQL và cho tất cả các hệ thống cơ sở dữ liệu hiện đại như MS SQL Server, IBM DB2, Oracle, MySQL và Microsoft Access.

2.1.1 Sơ lược Microsoft SQL Server

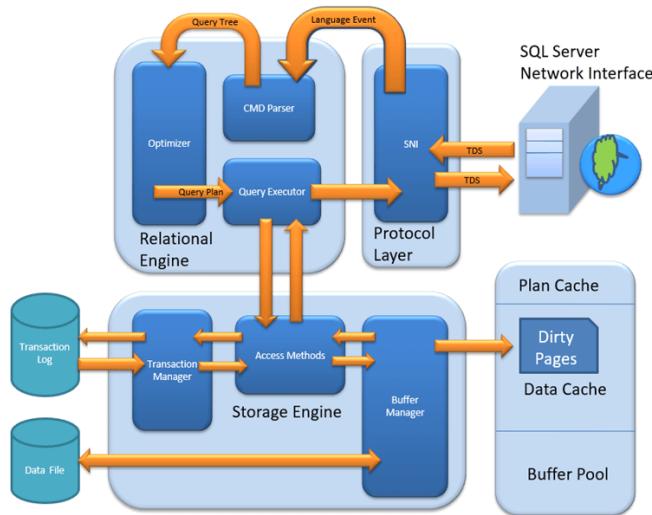
Microsoft SQL Server là một RDBMS được phát triển bởi Microsoft. Giống như phần mềm RDBMS khác, Microsoft SQL Server được xây dựng dựa trên cơ sở của SQL - một ngôn ngữ lập trình tiêu chuẩn mà các Database Administrators(DBAs) và các chuyên gia IT khác sử dụng để quản lý và truy vấn dữ liệu.



Hình 2.1: MSSQL Server Introduction

2.1.2 Kiến trúc Microsoft SQL Server

Microsoft SQL Server sử dụng kiến trúc Client - Server cho các kết nối đến hệ cơ sở dữ liệu từ dịch vụ bên ngoài bao gồm các ứng dụng, các tiến trình(Process) trên cùng một máy chủ hoặc từ một máy tính khác gửi yêu cầu vào máy chủ. Quá trình MS SQL Server bắt đầu bằng việc ứng dụng khách(Guest) gửi yêu cầu. SQL Server chấp nhận, xử lý và trả lời yêu cầu với dữ liệu đã được xử lý.



Hình 2.2: MSSQL Server Architecture

Như Sơ đồ bên trên mô tả, có ba thành phần chính trong Kiến trúc của Microsoft SQL Server:

- Protocol Layer – SNI
- Relational Engine
- Storage Engine

Protocol Layer – SNI:

Đây là Layer giúp Microsoft SQL Server có thể giao tiếp với môi trường bên ngoài. Nó được gọi là SQL Server Network Interface và chứa các giao thức Microsoft SQL Server. Đây là các giao thức chính hiện được liên kết với SQL Server như Shared Memory, Named Pipes, TCP/IP và TDS (Tabular Data Stream). Ngoài ra, Microsoft SQL Server còn có giao thức Virtual Interface Adapter (VIA). Tuy nhiên, nó yêu cầu phải thiết lập Auxiliary Hardware và Microsoft không dùng nữa, hiện tại thì không còn có sẵn trong các phiên bản SQL Server mới nhất.

Shared Memory là giao thức đơn giản nhất, được sử dụng mặc định cho các kết nối cục bộ, khi ứng dụng Client và SQL Server được đặt trên cùng một máy.

Named Pipes là một giao thức có thể được sử dụng trong trường hợp ứng dụng Client và SQL Server được kết nối qua mạng cục bộ. Theo mặc định, giao thức này bị tắt, chúng ta có thể kích hoạt nó bằng SQL Configuration Manager với Port 445.

TCP/IP là giao thức quan trọng nhất trong nhóm, đây là giao thức chính kết nối với SQL Server từ xa, sử dụng IP Address và số Port. Vì vậy khi ứng dụng Client và SQL Server được cài đặt trên các máy riêng biệt. Port TCP mặc định được SQL Server sử dụng là 1433. Tuy nhiên, chúng ta cũng nên thay đổi cổng sau khi cài đặt Microsoft SQL Server.

Tabular Data Stream là một giao thức ở mức Application-Level được áp dụng để truyền yêu cầu và phản hồi giữa các ứng dụng Client và SQL Server Systems. Client thường thiết lập kết nối lâu dài với



Server. Ngay sau khi kết nối được thiết lập, các thông báo TDS sẽ trở thành phương tiện liên lạc giữa ứng dụng Client và Database Server. Ngoài ra, Server cũng có thể hoạt động như một Client với yêu cầu kết nối TDS riêng.

Relational Engine:

Relational Database Engine trong Microsoft SQL Server còn được gọi là Query Processor vì nó xác định việc thực hiện các truy vấn cũng như tiến hành quản lý bộ nhớ, quản lý luồng, các tác vụ cũng như quản lý bộ đệm. Nó yêu cầu dữ liệu từ Storage Engine và xử lý kết quả để trả về cho người dùng. Relational Engine bao gồm 3 thành phần chính: Query Parser, Optimizer, and Query Executor.

Query Parser là thành phần nhận truy vấn, kiểm tra các lỗi cú pháp và ngữ nghĩa và cuối cùng là xây dựng cấu trúc cây từ dạng văn bản của truy vấn, cụ thể.

Kiểm tra cú pháp truy vấn đầu vào của người dùng xem nó có đúng cú pháp hay không. Nếu truy vấn không tuân thủ cú pháp SQL thì Query Parser sẽ trả về thông báo lỗi.

Kiểm tra ngữ nghĩa là phân tích xem bảng và cột được truy vấn có tồn tại trong lược đồ cơ sở dữ liệu hay không. Nếu kiểm tra thành công, bảng được truy vấn sẽ được liên kết với truy vấn. Nếu bảng/cột không tồn tại, Query Parser sẽ trả về thông báo lỗi. Độ phức tạp của việc kiểm tra ngữ nghĩa phụ thuộc vào độ phức tạp của truy vấn. Cuối cùng, Query Parser tạo cây thực thi cho truy vấn.

Optimizer tiến hành tạo kế hoạch thực thi rẻ nhất cho truy vấn, nghĩa là giảm thiểu thời gian chạy của nó càng nhiều càng tốt. Việc tính toán chi phí của chúng dựa trên mức sử dụng CPU và bộ nhớ cũng như nhu cầu đầu vào/đầu ra. Việc tối ưu hóa được Optimizer tiến hành qua 3 giai đoạn: Trivial Plan, Transaction Processing Plan và Parallel Processing/Optimization.

Cuối cùng, Query Executor gọi phương thức truy cập. Sau khi dữ liệu cần thiết được lấy từ Storage Engine, nó sẽ chuyển đến Server Network Interface ở Potocol Layer và được gửi đến người dùng.

Storage Engine:

Storage Engine xử lý việc lưu trữ và truy xuất dữ liệu khi người dùng yêu cầu. Nó chứa Buffer Manager và Transaction Manager tương tác với dữ liệu và Log Files theo truy vấn.

Access Method sẽ xác định xem truy vấn của người dùng có chứa câu lệnh SELECT hay non-SELECT hay không. Nếu chứa SELECT, truy vấn được chuyển đến Buffer Manager để xử lý. Nếu là non-SELECT thì nó được chuyển đến Transaction Manager.

Buffer Manager chịu trách nhiệm phân vùng bộ nhớ chính khả dụng thành các bộ đệm, đây là các vùng có kích thước trang mà khối đĩa(Disk Block) có thể được chuyển vào.

Transaction Manager được gọi khi truy vấn bao gồm câu lệnh non-SELECT. Nó quản lý giao dịch bằng Log Manager và Lock Manager.

Log Manager theo dõi tất cả các bản cập nhật trong hệ thống bằng Transaction Logs, Log có Số thứ tự với Transaction ID và Data Modification Record giúp theo dõi Transaction Committed và Transaction Rollback.

Trong suốt quá trình giao dịch, Lock Manager đảm bảo dữ liệu liên quan nằm trong Data Storage thì luôn ở trạng thái Khóa và nó cũng đảm bảo tuân thủ các thuộc tính ACID.

2.1.3 Microsoft SQL Server hiện tại

Microsoft SQL Server hiện tại đạt được danh tiếng về kiến trúc, độ tin cậy, bộ tính năng mạnh mẽ, khả năng mở rộng và sự cống hiến của cộng đồng mã nguồn mở đãng sau phần mềm để cung cấp các giải pháp sáng tạo và hiệu quả một cách nhất quán.Thêm vào đó, Microsoft SQL Server cung cấp khả năng



lưu trữ dữ liệu hiệu suất cao. Chúng giúp quản lý các tập dữ liệu khổng lồ trên mọi máy tính khi được kết nối với mạng.

SQL Server đã tồn tại được hơn ba thập kỷ và trải qua một vài Versions từ năm 1989 khi còn là Microsoft And Sybase Released Version 1.0 đến năm 2019 Microsoft Releases SQL Server 2019, Introducing Big Data Clusters với các bản Edition phổ biến trên thị trường như: SQL Server Enterprise, SQL Server Standard, SQL Server WEB, SQL Server Developer. Không có gì ngạc nhiên khi Microsoft SQL Server đã trở thành cơ sở dữ liệu quan hệ nguồn mở được nhiều người và tổ chức lựa chọn.

2.2 Indexing

2.2.1 Giới thiệu

Mục đích của Index, ở trong SQL Server hay không, về cơ bản là để truy vấn dữ liệu được nhanh và hiệu quả nhờ vào giảm thiểu việc quét toàn bộ bảng dữ liệu để lọc các hàng dữ liệu. Tuy vậy, Index không phải hữu dụng trong tất cả các câu truy vấn. Đôi khi việc sử dụng Index sẽ là kém hiệu quả và Query Planner sẽ thay thế bằng một phương án khác, kể cả là việc quét toàn bộ bảng thông thường.

2.2.2 Index trong Microsoft SQL Server

SQL Server cung cấp nhiều loại Index: Hash, Memory-Optimized Nonclustered, Clustered, Nonclustered và Unique. Mỗi loại sử dụng một phương pháp, thuật toán khác nhau để hiện thực. Ta sẽ nói đến một vài cơ chế Index đặc biệt sau đây.

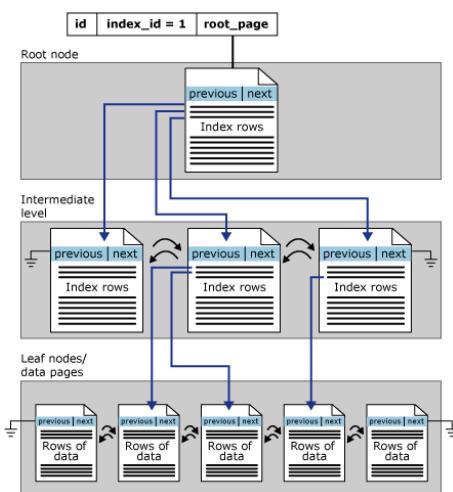
2.2.2.1 Clustered

Clustered Indexes sẽ sắp xếp và lưu trữ các hàng dữ liệu trong bảng dựa trên các Key Values của chúng. Các Key Values này là các cột có trong khi định nghĩa Index. Chỉ có thể có một Clustered Index trên mỗi bảng vì bản thân các hàng dữ liệu chỉ có thể được lưu trữ theo một cột có thứ tự.

Duy nhất các hàng dữ liệu trong bảng được lưu trữ theo thứ tự được sắp xếp là khi bảng đó có chứa Clustered Index. Khi một bảng có Clustered Index, bảng đó được gọi là Clustered Table. Nếu một bảng không có Clustered Index, các hàng dữ liệu của nó được lưu trữ trong một cấu trúc không có thứ tự gọi là Heap.

Rowstore Indexes được tổ chức dưới dạng B+ Trees. Mỗi trang trong Index B+ Tree được gọi là Index Node. Node trên cùng của Index được gọi là Root Node. Các Node dưới cùng trong Index được gọi là Leaf Nodes. Bất kỳ Index Levels nào giữa Root Node và Leaf Nodes đều được gọi chung là Intermediate Levels. Trong một Clustered Index, Leaf Nodes chứa các trang dữ liệu của bảng. Còn Root và Intermediate Level Nodes chứa các trang Index chứa Index Rows. Mỗi Index Rows chứa một giá trị khóa và một con trỏ tới trang Intermediate Level trong B+ Tree hoặc hàng dữ liệu ở Leaf Level của Index. Ngoài ra, các trang ở mỗi Level Index được liên kết trong một Doubly Linked List.

Hình bên dưới cho thấy cấu trúc của một Clustered Indexes trong một phân vùng duy nhất.



Hình 2.3: Cấu trúc Clustered Index

Ví dụ, chúng tôi hãy đi vào chi tiết về cách tìm kiếm một bản ghi trong bảng theo Clustered Indexes. Đầu tiên, chúng tôi sẽ tạo một bảng Cars và sau đó sẽ Insert một số dữ liệu mẫu.

```
1 CREATE Table Cars (Id INT, BrandName VARCHAR(100))
2 GO
3 INSERT INTO Cars VALUES(1,'Ford')
4 INSERT INTO Cars VALUES(2,'Fiat')
5 INSERT INTO Cars VALUES(3,'Mini')
6 INSERT INTO Cars VALUES(4,'Jaguar')
7 INSERT INTO Cars VALUES(5,'Kia')
8 INSERT INTO Cars VALUES(6,'Nissan')
9 INSERT INTO Cars VALUES(7,'BMW')
10 INSERT INTO Cars VALUES(8,'Mercedes')
11 INSERT INTO Cars VALUES(9,'Mazda')
12 INSERT INTO Cars VALUES(10,'Volvo')
13 INSERT INTO Cars VALUES(11,'Lexus')
14 INSERT INTO Cars VALUES(12,'Buick')
15 INSERT INTO Cars VALUES(13,'GMC')
16 INSERT INTO Cars VALUES(14,'Honda')
17 INSERT INTO Cars VALUES(15,'Lotus')
18 INSERT INTO Cars VALUES(16,'Opel')
19 INSERT INTO Cars VALUES(17,'Bentley')
20 INSERT INTO Cars VALUES(18,'Dodge')
21 INSERT INTO Cars VALUES(19,'Tesla')
22 INSERT INTO Cars VALUES(20,'Porche')
23 INSERT INTO Cars VALUES(21,'Ferrari')
24 INSERT INTO Cars VALUES(22,'Audi')
25 GO
```

Xác định một Unique Clustered Index cho bảng Cars:

```
1 CREATE UNIQUE CLUSTERED INDEX IX_001
2 ON Cars (Id);
```

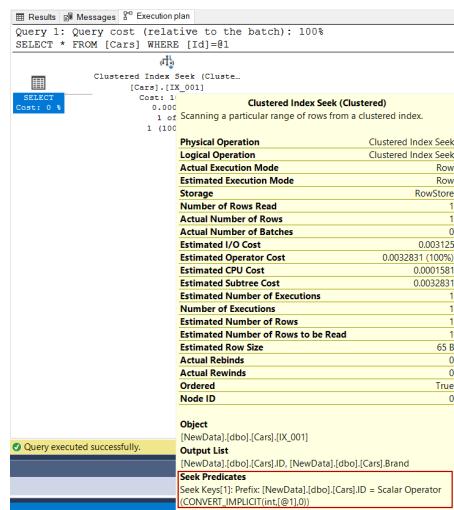
SQL Server thực hiện quy trình tìm kiếm Clustered Index khi truy cập dữ liệu bằng cấu trúc B-tree và được thể hiện bằng biểu tượng sau trong Execution Plans:



Hình 2.4: Biểu tượng quy trình tìm kiếm Clustered Index

Khi chúng tôi thực hiện truy vấn sau, nó sẽ thực hiện thao tác tìm kiếm Clustered Index. Hoạt động tìm kiếm Clustered Index sử dụng cấu trúc của B-Tree rất hiệu quả và dễ dàng tìm thấy (các) hàng đủ điều kiện:

```
1 SELECT *
2 FROM Cars
3 WHERE Id = 12;
```



Hình 2.5: Kết quả Clustered Index Seek

Trong Execution Plan trên, Seek Predicates chỉ ra rằng Storage Engine sử dụng cấu trúc B-Tree và tìm Leaf Level lưu trữ các hàng dữ liệu. Đối với truy vấn trên, tính duy nhất của Clustered Index là rất quan trọng vì ràng buộc này đảm bảo rằng chỉ một hàng sẽ được trả về từ truy vấn. Khái niệm tìm kiếm dữ liệu này được gọi là Singleton Seek.

Trong trường hợp, nếu Clustered Indexes không được tạo bằng thuộc tính UNIQUE thì Database Engine sẽ tự động thêm cột 4-byte Uniqueifier vào bảng. Khi được yêu cầu, Database Engine sẽ tự động thêm Uniqueifier vào một hàng để làm cho mỗi khóa trở thành duy nhất. Cột này và các giá trị của nó được sử dụng nội bộ và người dùng không thể xem hoặc truy cập được.

Ví dụ, chúng tôi sẽ tạo một bảng, nó chỉ bao gồm hai cột và sau đó chúng ta tạo Non-Unique Clustered Index cho bảng này.

```
1 CREATE TABLE TestDuplicate
2 (
3     IdNumber INT,
4     Col1      VARCHAR(100));
5 GO
6 CREATE CLUSTERED INDEX IX_001
7 ON TestDuplicate
8 (IdNumber);
```

Bây giờ, chúng tôi sẽ chèn một hàng vào bảng này và truy vấn kích thước bản ghi tối đa và tối thiểu của các trang Index dùng sys.dm_db_index_physical_stats.

```
1 INSERT INTO TestDuplicate
2 (IdNumber,
3     Col1)
4 VALUES
5 (
6     1, 'Text-1');
7
8 SELECT min_record_size_in_bytes,
9       max_record_size_in_bytes
10  FROM sys.dm_db_index_physical_stats
11 (DB_ID(), OBJECT_ID('TestDuplicate'), NULL, NULL, N'SAMPLED');
```

Results		Messages
	min_record_size_in_bytes	max_record_size_in_bytes
1	23	23

Hình 2.6: Kết quả truy vấn kích thước bản ghi min-max

Thêm một hàng trùng lặp và kiểm tra lại cột kích thước bản ghi tối đa.

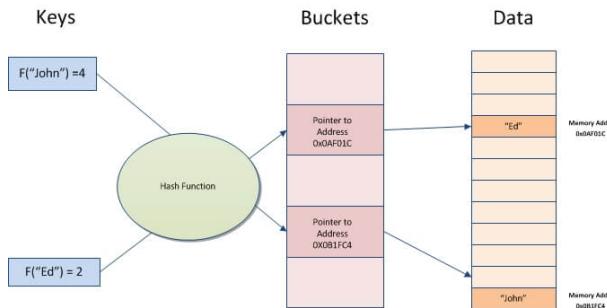
```

1 INSERT INTO TestDuplicate
2   (IdNumber,
3    Col1)
4 VALUES
5 (
6     1, 'Text-2');
7
8 SELECT min_record_size_in_bytes,
9       max_record_size_in_bytes
10  FROM sys.dm_db_index_physical_stats
11  (DB_ID(), OBJECT_ID('TestDuplicate'), NULL, NULL, N'SAMPLED');
```

Results		Messages
	min_record_size_in_bytes	max_record_size_in_bytes
1	23	27

Hình 2.7: Kết quả kiểm tra lại kích thước bản ghi max

2.2.2.2 Hash



Hình 2.8: Kiến trúc Hash

Hash Index bao gồm một mảng các con trỏ và mỗi phần tử của mảng được gọi là Hash Bucket. Mỗi Bucket có 8 byte, được sử dụng để lưu trữ địa chỉ bộ nhớ của danh sách liên kết các Key Entries. Mỗi Key Entry là một giá trị cho một Index Key, cộng với địa chỉ của hàng tương ứng trong bảng được tối ưu hóa bộ nhớ. Mỗi entry trả đến Entry tiếp theo trong danh sách liên kết các Entries, tất cả đều được nối với Bucket hiện tại.

```

1 ALTER TABLE customers
2 ADD INDEX idx_customers_id_hash
3 HASH (id) WITH (BUCKET_COUNT = 1000);
```

Trong ví dụ trên, "idx_customers_id_hash" là tên chúng tôi đặt cho Index, "customers" là tên của bảng và "id" là tên của cột chúng tôi đang lập Index. Tham số "BUCKET_COUNT = 1000" được sử dụng để chỉ định số lượng nhóm mà Index cần có. Điều này rất quan trọng vì nó xác định số lượng phân vùng mà Index sẽ được chia thành và do đó ảnh hưởng đến tốc độ truy vấn.

Hash Index cần số lượng Buckets phải được chỉ định tại thời điểm tạo Index với số lượng Buckets tối đa là 1.073.741.824. Ưu điểm chính của việc sử dụng Hash Index là nó có thể mang lại hiệu suất tìm kiếm rất nhanh cho các truy vấn khớp chính xác, đặc biệt khi cột được lập Index có nhiều giá trị riêng biệt. Tuy nhiên, Hash Index không hỗ trợ truy vấn hoặc sắp xếp phạm vi và chúng cũng có thể tiêu tốn một lượng lớn bộ nhớ và dung lượng ổ đĩa.

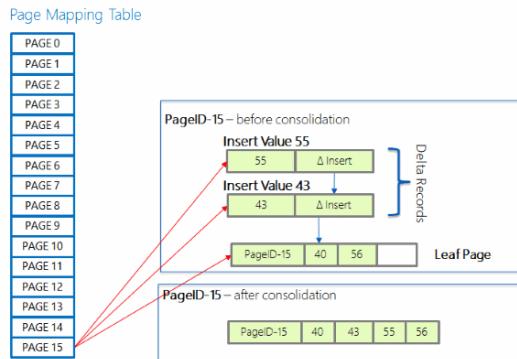
2.2.2.3 Memory-Optimized Nonclustered

In-memory Nonclustered Indexes được triển khai bằng cách sử dụng cấu trúc dữ liệu có tên là Bw-Tree, được Microsoft Research ban hành ban đầu vào năm 2011. Bw-Tree là một biến thể của B-Tree.

Ở cấp độ cao, Bw-Tree có thể được hiểu là Map các trang được sắp xếp theo Page ID(PidMap), một phương tiện để phân bổ và sử dụng lại Page ID(PidAlloc) và một tập hợp các trang được liên kết trong Page Map. Ba thành phần này tạo nên cấu trúc bên trong cơ bản của Bw-Tree.

Có 3 cách khác nhau có để quản lý cấu trúc của Bw-Tree: Consolidation, Split, and Merge.

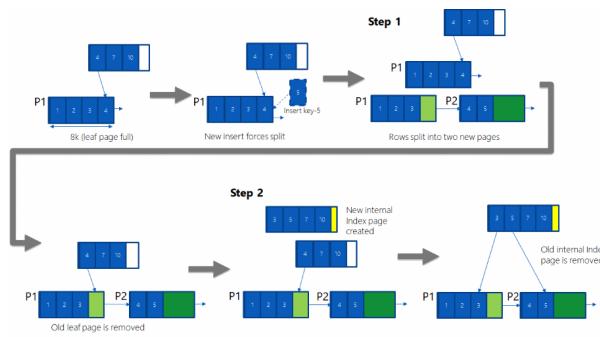
Delta Consolidation: một chuỗi dài Delta Records có thể làm giảm hiệu suất tìm kiếm vì điều này có thể giống với việc chúng ta đang đi qua các chuỗi dài khi tìm kiếm thông qua một Index. Nếu một Delta Record mới được thêm vào một chuỗi đã có 16 phần tử, các thay đổi trong các bản ghi delta sẽ được hợp nhất vào Index Page được tham chiếu và sau đó trang sẽ được xây dựng lại, bao gồm các thay đổi được chỉ ra bởi Delta Record. Trang mới được xây dựng lại sẽ có cùng Page ID nhưng có địa chỉ bộ nhớ mới.



Hình 2.9: Delta Records

Split Page: Page Index trong Bw-tree sẽ phát triển theo nhu cầu, bắt đầu từ việc lưu trữ một hàng đơn lẻ đến lưu trữ tối đa 8 KB. Khi Page Index tăng lên 8 KB, việc chèn một hàng mới sẽ khiến Page Index bị chia tách. Đối với một Internal Page, việc này có nghĩa là khi không còn chỗ để thêm một Key Value và con trỏ, còn đối với một Leaf Page thì số hàng sẽ quá lớn để vừa với trang sau khi tất cả Delta Records được tích hợp. Thông tin thống kê trong Page Header cho Leaf Page sẽ theo dõi lượng không gian cần thiết để hợp nhất Delta Records. Thông tin này được điều chỉnh khi mỗi Delta Record mới được thêm vào.

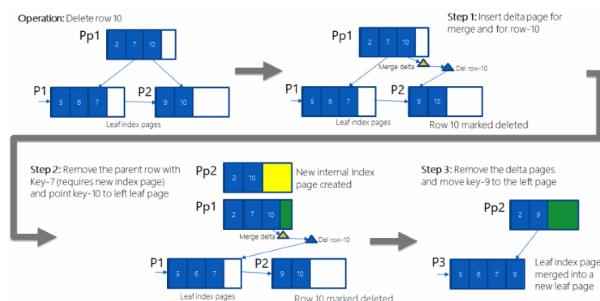
Một hoạt động phân chia được thực hiện theo hai bước. Hình bên dưới, giả sử Leaf Page buộc phải phát triển vì một khóa có giá trị 5 đang được chèn và Nonleaf Page tồn tại đang trỏ đến cuối leaf-level page. (Key Value 4).



Hình 2.10: Split Page

Merge: khi thực hiện DELETE dẫn đến một trang có kích thước nhỏ hơn 10% kích thước trang tối đa (hiện tại là 8 KB), trang đó sẽ được hợp nhất với một trang liền kề. Khi một hàng bị xóa khỏi một trang, Delta Record sẽ được thêm vào để xóa . Ngoài ra, việc kiểm tra được thực hiện để xác định xem Index Page có đủ điều kiện để Merge hay không. Kiểm tra này xác minh xem khoảng trống còn lại sau khi xóa hàng có nhỏ hơn 10% kích thước trang tối đa hay không. Nếu nó đủ điều kiện, việc Merge được thực hiện theo ba bước cơ bản.

Hình bên dưới, giả sử thao tác DELETE sẽ xóa Key Value có giá trị 10.



Hình 2.11: Merge Page

2.2.2.4 Non-clustered

Non-clustered Index chứa các giá trị Index Key và con trỏ đến vị trí lưu trữ của dữ liệu bảng. Bạn có thể tạo nhiều Non-clustered Index trên một bảng. Nói chung, các Non-clustered Index phải được thiết kế để cải thiện hiệu suất của các câu truy vấn được sử dụng thường xuyên mà không nằm trong Clustered Index.

Tương tự như cách chúng ta sử dụng mục lục trong sách, Query Optimizer sẽ tìm kiếm dữ liệu bằng cách tìm kiếm Non-clustered Index để tìm vị trí của dữ liệu trong bảng rồi truy xuất dữ liệu trực tiếp từ vị trí đó. Điều này làm cho các Non-clustered Index trở thành lựa chọn tối ưu cho các truy vấn khớp chính xác.

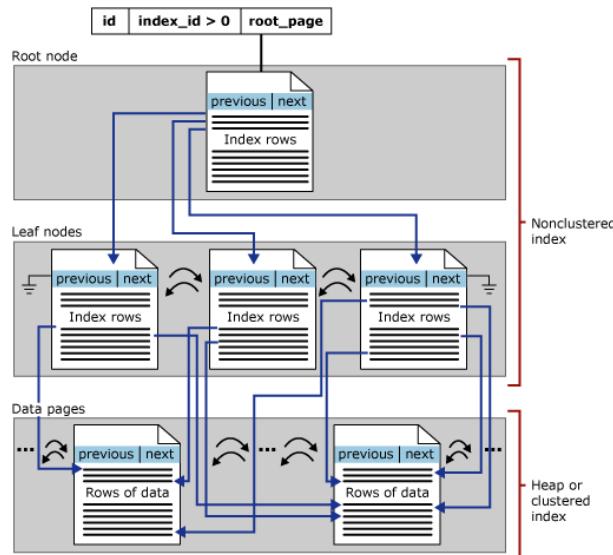
```

1 -- Tạo Non-clustered Index trên cột ManagerID
2 CREATE NONCLUSTERED INDEX IX_Employee_ManagerID
3 ON Employee (ManagerID);
4 -- Truy vấn sử dụng Non-clustered Index
5 SELECT *
6 FROM Employee
7 WHERE ManagerID = 1;

```

Ví dụ trên ta thấy: giả sử để truy vấn bảng HumanResources.Employee cho tất cả nhân viên báo cáo cho một người quản lý cụ thể có ID = 1, Query Optimizer có thể sử dụng Non-clustered Index là IX_Employee_ManagerID, cột này có ManagerID làm cột chính. Query Optimizer có thể nhanh chóng tìm thấy tất cả Entries trong Index khớp với ManagerID đã chỉ định. Mỗi Index Entry trả đến trang và hàng chính xác trong bảng, nơi dữ liệu tương ứng có thể được tìm thấy. Sau khi tìm thấy tất cả Entries trong Index, Query Optimizer truy cập trực tiếp vào trang và hàng chính xác để truy xuất dữ liệu.

Hình bên dưới cho thấy cấu trúc của một Non-clustered Index trong một phân vùng.



Hình 2.12: Kiến trúc Non-clustered Index

Non-clustered Index có cấu trúc B+Tree giống như các Clustered Index. Tuy nhiên, có một số khác biệt quan trọng. Đầu tiên, các hàng dữ liệu của bảng bên dưới không được sắp xếp và lưu trữ theo thứ tự theo các Non-clustered Key của chúng. Thứ hai, Leaf Level của Non-clustered Index được tạo thành từ Index Pages thay vì Data Pages. Index Pages ở Leaf Level chứa các cột chính và các cột được Included.

Không giống Clustered Index, chúng ta có thể tạo nhiều Non-clustered Index trên một bảng. Tuy nhiên, chúng ta cần nhớ rằng bạn càng có nhiều Non-clustered Index thì càng có nhiều dữ liệu được sao chép. Nếu dữ liệu đó cần thay đổi, SQL Server cần đảm bảo các bản sao trong Non-clustered Index cũng được cập nhật.

Điều này đặc biệt đúng khi chúng ta bắt đầu INCLUDE các cột trong Non-clustered Index. Tất nhiên, các truy vấn của mình sẽ hơn nếu ban INCLUDE tất cả hoặc hầu hết các cột trong bảng. Vì vậy, sẽ ra kết quả rất nhanh vì SQL Server sẽ không bao giờ cần phải tra cứu. Tuy nhiên, việc này rất có thể sẽ làm giảm hiệu suất ghi vì dữ liệu đã sao chép sẽ cần được cập nhật sau bất kỳ bản sửa đổi nào.

Lưu ý rằng SQL Server không cho INCLUDE các cột khi chúng ta tạo Non-clustered Index trong câu lệnh CREATE TABLE. Ví dụ: đây là câu lệnh CREATE TABLE cho bảng có tên Books2 trong đó chúng tôi tạo Non-clustered Index trên cột Title nhưng không thể INCLUDE cột khác đến Index:

```
1 CREATE TABLE Books2
2 (
3     BookID INT,
4     Title VARCHAR(50) INDEX idx_BookTitle NONCLUSTERED INCLUDE Pages,
5     Author VARCHAR(15),
6     Pages INT
7 )
```



Chúng tôi sẽ nhận được thông báo lỗi nếu có chạy mã trên.

2.2.2.5 Unique

Unique Index là một Index có khóa duy nhất. Khóa duy nhất là khóa bao gồm một cột trong bảng hoặc một nhóm các cột trong bảng có các giá trị duy nhất hoặc tổ hợp các giá trị duy nhất tương ứng. Một Unique Index được tạo tự động ở chế độ nền khi thực thi ràng buộc duy nhất trên một cột trong bảng. Nếu không, nó phải được tạo thủ công bằng cách sử dụng cột trong bảng hoặc nhóm cột chỉ có các giá trị duy nhất làm khóa Index Key. Nếu một cột hoặc bất kỳ cột nào trong nhóm được chọn làm Index Key có các giá trị trùng lặp thì việc tạo Unique Index sẽ không thành công. Vì vậy, sau khi tạo thành công, Unique Index sẽ không cho phép chèn các giá trị trùng lặp vào cột khóa hoặc nhóm các cột khoa.

Unique Index có thể được clustered hoặc non-clustered. Nếu Primary Key trong bảng có ràng buộc duy nhất được thực thi trên đó thì Unique Index mặc định được tạo trên Primary Key sẽ là Unique Clustered Index. Như tất cả các Index khác, mục đích của việc tạo một Unique Index trên một bảng là cải thiện hiệu suất tìm kiếm và truy vấn bằng cách giảm số lần quét và thay thế chúng bằng các tìm kiếm.

Ví dụ, để kiểm tra tính Unique, chúng tôi sẽ cố gắng tạo một Unique Non-Clustered Index trên bảng bằng cách sử dụng cột customer_email làm Index Key. Lệnh bên dưới được thực hiện và gặp lỗi vì có giá trị trùng lặp trong cột customer_email. Cho nên Microsoft SQL Server sẽ không cho phép tạo Unique Index trên khóa dựa trên cột có giá trị trùng lặp.

```
1 CREATE UNIQUE INDEX
2 ix_custom_unique
3 ON
4 customers(customer_email);
5
6 Msg 1505, Level 16, State 1, Line 4
7 The CREATE UNIQUE INDEX statement terminated because a duplicate key was found
8 for the object name 'dbo.customers' and the index name 'ix_custom_unique'. The
9 duplicate key value is (lblair90@yahoo.com).
10 The statement has been terminated.
```

Vì vậy, để có thể tạo Unique Index, chúng ta phải làm cho tất cả các giá trị cột là Unique, bằng cách cập nhật giá trị customer_email đối với khách hàng tương ứng.

Unique Index mang lại nhiều lợi ích. Đầu tiên, với điều kiện là dữ liệu trong mỗi cột là Unique, bạn có thể tạo cả Unique Clustered Index và Multiple Unique Nonclustered Indexes trên cùng một bảng. Thứ hai, Các Unique Index đảm bảo tính toàn vẹn dữ liệu của các cột được xác định.Thêm vào đó, Unique Index giúp cung cấp thông tin bổ sung hữu ích cho trình tối ưu hóa truy vấn có thể tạo ra các kế hoạch thực thi hiệu quả hơn.

2.2.2.6 Columnstore

Columnstore là loại Index chuyên cho báo cáo và phân tích dữ liệu, cung cấp khả năng tối ưu tốc độ xử lý truy vấn cho các câu lệnh truy vấn cũng như những Data Warehousing Workload. Trong nhiều trường hợp, tốc độ truy vấn vào kho dữ liệu tăng nhanh từ hàng chục đến hàng trăm lần.

Xét về mặt cấu trúc dữ liệu, trước đây, dữ liệu trong SQL Server được lưu trữ trong bảng dưới dạng Heap hay Index. Trong cả 2 cách lưu trữ này, SQL Server lưu trữ dữ liệu trong các Page theo dạng Row-Based, tức là tất cả các giá trị trong 1 dòng dữ liệu (trên nhiều cột) đều được lưu trữ trong 1 Page. Các lưu trữ này thông thường gọi là Row-Store.

Đối với Columnstore Index thì tất cả các giá trị trên 1 Column sẽ được lưu trữ tuần tự trong 1 tập các Page gọi là Column-Store và có thể ví như chúng ta làm 1 hành động xoay cách lưu trữ truyền thống Row-Store 1 góc 90 độ. Ví dụ:



```
1 SET STATISTICS IO ON;
2 SET STATISTICS TIME ON;
3 SELECT bp.Name AS ProductName,
4 COUNT(bth.ProductID),
5 SUM(bth.Quantity),
6 AVG(bth.ActualCost)
7 FROM dbo.bigProduct AS bp
8 JOIN dbo.bigTransactionHistory AS bth
9 ON bth.ProductID = bp.ProductID
10 GROUP BY bp.Name
11 OPTION (RECOMPILE);
```

Trong trường hợp trên, chúng tôi có bảng dbo.bigProduct có 25K Row, bảng dbo.bigTransactionHistory có 31M Row. Khi chúng tôi chạy thì kết quả trả về với tổng thời gian query là 8s(7933ms). SQL Server phải scan tất cả các Row của cả 2 bảng để lấy dữ liệu, đọc 131888 Page của bảng bigTransactionHistory và 603 Page của bảng bigProduct(cho phép JOIN) từ RAM.

Chúng tôi sẽ thêm Columnstore Index:

```
1 CREATE NONCLUSTERED COLUMNSTORE INDEX ix_csTest
2 ON dbo.bigTransactionHistory
3 (
4 ProductID,
5 Quantity,
6 ActualCost
7 );
```

Chúng tôi kiểm tra lại kết quả và thấy rằng tổng thời gian Query là 449ms(0.5s). Đọc ở bảng bigProduct là 622 Page(so với cũ là 603) không khác nhau nhiều. Đọc ở bảng bigTransactionHistory là 0 nhưng xuất hiện thêm thông tin mới(Segment Reads = 33). Quan trọng nhất là thời gian Query đã giảm từ 8s xuống 0.5s tức là tăng 16 lần. Như vậy, rõ ràng ColumnStore index dùng cực kỳ hiệu quả trong trường hợp này.

Tuy nhiên, việc dùng ColumnStore Index lại ít phổ biến vì khi dùng ColumnStore Index cho 1 bảng, chúng ta không thể Update bảng đó, trừ khi Remove Index này. Điều này khiến cho ColumnStore không thể sử dụng trên các hệ thống OLTP, tức là các hệ thống có dữ liệu update liên tục(như ta vẫn thường sử dụng).

2.2.2.7 Một số loại Index khác

Filtered

Filtered Index là Non-Clustered Index được tối ưu hóa, phù hợp cho các câu truy vấn SELECT từ một tập hợp dữ liệu con được xác định. Filtered Index sử dụng Filter Predicate để lập Index vào một phần của một hàng trong bảng. Filtered Index được thiết kế tốt có thể cải thiện hiệu suất truy vấn, giảm chi phí bảo trì Index và giảm chi phí lưu trữ Index so với Full-Table Indexes.

Full-Text

Full-Text Index là một loại Index đặc biệt cung cấp quyền truy cập Index cho các truy vấn văn bản đối với dữ liệu cột là ký tự hoặc nhị phân. Full-Text Index chia cột thành các Tokens, các Tokens này tạo nên dữ liệu Index. Trước khi tạo Full-Text Index, cần tạo FULL TEXT CATALOG là nơi lưu trữ dữ liệu Full-Text Index. FULL TEXT CATALOG có thể chứa nhiều Index nhưng Full-Text Index chỉ có thể là một phần của một CATALOG. Ngoài ra, chỉ có thể tạo một Full-Text Index trên mỗi bảng, nếu bạn có nhiều cột, thì các cột đó phải được chuyển sang các bảng riêng. Full-Text Index không được cập nhật ngay lập tức như trường hợp của các Index thông thường. Việc điều chỉnh Full-Text Index có thể tốn nhiều tài nguyên nên có nhiều tùy chọn hơn cho phép bạn kiểm soát thời điểm chúng được cập nhật như: Full population;



Automatic or manual population based on change tracking; Incremental population based on a timestamp.

XML

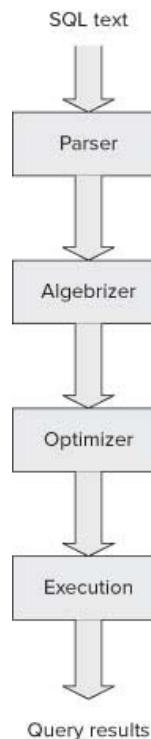
XML Index có thể được tạo trên các cột kiểu dữ liệu là XML. Chúng lập Index cho tất cả các thẻ, giá trị và đường dẫn trên các phiên bản XML trong cột và mang lại lợi ích cho hiệu suất truy vấn. Có hai loại XML Index khác nhau: Primary và Secondary. XML Index trong các tình huống: Đầu tiên, truy vấn trên các cột XML là vấn đề thường xuyên trong công việc nên XML Index sẽ duy trì chi phí trong quá trình sửa đổi dữ liệu. Thứ hai, khi giá trị XML tương đối lớn và các phần được truy xuất tương đối nhỏ thì việc xây dựng Index sẽ tránh phân tích cú pháp cho toàn bộ dữ liệu trong khi Run Time và mang lại lợi ích khi tra cứu Index để xử lý hiệu quả truy vấn.

2.3 Query processing và optimization

2.3.1 Query processing

Việc xử lý truy vấn được thực hiện bởi Relational Engine trong Microsoft SQL Server. Đó là quá trình lấy các câu lệnh T-SQL chúng ta viết và chuyển đổi chúng thành thứ có thể đưa ra yêu cầu tới Storage Engine và truy xuất các kết quả cần thiết.

Microsoft SQL Server thực hiện 4 bước để xử lý một truy vấn: phân tích cú pháp(Parsing), đại số hóa(Algebrizing), tối ưu hóa(Optimizing) và thực thi(Execution), chúng được hiển thị như hình bên dưới.



Hình 2.13: 4 bước xử lý truy vấn trong Microsoft SQL Server

3 bước đầu tiên đều được thực hiện bởi Relational Engine. Kết quả của bước thứ 3 là kế hoạch tối ưu hóa được lên lịch và trong đó có các yêu cầu được thực hiện tới Storage Engine để truy xuất dữ liệu và trả kết quả dữ liệu mà chúng ta truy vấn.

2.3.1.1 Phân tích cú pháp(Parsing)

Trong giai đoạn phân tích cú pháp, SQL Server thực hiện các kiểm tra cơ bản về mã nguồn T-SQL. Trình phân tích cú pháp này tìm kiếm cú pháp SQL không hợp lệ, chẳng hạn như sử dụng sai các từ dành riêng, tên cột hoặc bảng,...

Nếu quá trình phân tích cú pháp hoàn tất mà không có lỗi, nó sẽ tạo ra một cây phân tích cú pháp, được chuyển sang giai đoạn liên kết, xử lý truy vấn tiếp theo. Cây phân tích cú pháp là một biểu diễn nội bộ của câu truy vấn. Nếu quá trình phân tích cú pháp phát hiện bất kỳ lỗi nào, quá trình sẽ dừng lại và lỗi sẽ được trả về.

2.3.1.2 Đại số hóa(Algebrizing)

Giai đoạn đại số hóa còn được gọi là giai đoạn ràng buộc. Trong các phiên bản đầu tiên của SQL Server, giai đoạn này được gọi là chuẩn hóa. Trong quá trình đại số hóa, SQL Server thực hiện một số thao tác



trên cây phân tích cú pháp và sau đó tạo cây truy vấn được chuyển tối ưu hóa truy vấn.

Các bước được thực hiện trong quá trình đại số hóa theo mô hình sau:

- Bước 1: Phân giải tên(Name resolution) - Xác nhận rằng tất cả các đối tượng được hiển thị bằng cách kiểm tra tên bảng và cột để đảm bảo rằng chúng tồn tại và người dùng có quyền truy cập vào chúng.
- Bước 2: Dẫn xuất kiểu(Type derivation) - Xác định loại cuối cùng cho mỗi nút trong cây phân tích cú pháp.
- Bước 3: Liên kết tổng hợp(Aggregate binding) - Xác định vị trí thực hiện bất kỳ tổng hợp nào.
- Bước 4: Liên kết nhóm(Group binding) - Liên kết mọi tập hợp với danh sách chọn thích hợp.

Lỗi cú pháp được phát hiện trong giai đoạn này. Nếu gặp lỗi cú pháp, quá trình tối ưu hóa sẽ tạm dừng và lỗi sẽ được trả về cho người dùng.

2.3.2 Query optimization

2.3.2.1 Giới thiệu

Tối ưu hóa câu truy vấn là quá trình sinh ra một kế hoạch thực thi(Execution Plan) tối ưu từ cây truy vấn. Microsoft SQL Server hỗ trợ giải pháp truy vấn dữ liệu cho kết quả nhanh và hiệu quả cao, thế nhưng, trong lúc làm việc, người dùng cũng có thể gặp một số vấn đề với hiệu suất của câu truy vấn. Vậy nên chúng ta cần có một số chiến lược để góp phần tối ưu hóa và nâng cao hiệu năng câu truy vấn.

2.3.2.2 Chiến lược tối ưu

OR in the Join Predicate/WHERE Clause Across Multiple Columns

SQL Server có thể Filter một tập dữ liệu một cách hiệu quả bằng cách sử dụng các Index thông qua mệnh đề WHERE hoặc bất kỳ tổ hợp Filters nào được phân tách bằng toán tử AND. Các thao tác này lấy dữ liệu và cắt nó thành các phần nhỏ hơn dần dần cho đến khi chỉ còn lại tập kết quả mong muốn.

OR là một câu chuyện khác. Vì nó mang tính bao hàm nên SQL Server không thể xử lý nó trong một thao tác duy nhất. Thay vào đó, mỗi thành phần của OR phải được đánh giá độc lập. Khi hoạt động tốn kém này được hoàn thành, kết quả có thể được nối và trả về bình thường.

Trường hợp OR hoạt động kém nhất là khi có nhiều cột hoặc bảng có liên quan. Chúng ta không chỉ cần đánh giá từng thành phần của mệnh đề OR mà còn cần đi theo đường dẫn đó thông qua Filters và bảng khác trong truy vấn. Ngay cả khi chỉ có một vài bảng hoặc cột tham gia, hiệu suất có thể trở nên cực kỳ tệ.

Đây là một ví dụ rất đơn giản về cách OR có thể khiến hiệu suất trở nên tồi tệ hơn nhiều so với những gì chúng ta tưởng tượng:

```
1 SELECT DISTINCT
2     PRODUCT.ProductID,
3     PRODUCT.Name
4 FROM Production.Product PRODUCT
5 INNER JOIN Sales.SalesOrderDetail DETAIL
6 ON PRODUCT.ProductID = DETAIL.ProductID
7 OR PRODUCT.rowguid = DETAIL.rowguid;
```

Chúng tôi sẽ kiểm tra cả ProductID và rowguid thông qua việc thực hiện phép JOIN giữa hai bảng. Ngay cả khi không có cột nào trong hai cột này được lập Index, dự kiến là sẽ quét toàn bộ bảng Product và bảng SalesOrderDetail. Mặc dù có chi phí cao, nhưng ít nhất là điều này giúp chúng tôi hiểu được cách truy vấn được thực hiện. Tuy nhiên, kết quả thực thi trả về lại rất tốn kém, việc xử lý OR tốn rất nhiều



sức mạnh tính toán. 1,2 triệu lượt đọc đã được thực hiện! Xét rằng Product chỉ chứa 504 Records và SalesOrderDetail chứa 121317 Records, SQL Server đọc được nhiều dữ liệu hơn nội dung của mỗi bảng này. Ngoài ra, truy vấn mất khoảng 2 giây để thực thi trên máy tính để bàn chạy SSD.

Thật vậy, cách tốt nhất để xử lý OR là loại bỏ nó (nếu có thể) hoặc chia nó thành các truy vấn nhỏ hơn. Việc chia một truy vấn ngắn và đơn giản thành một truy vấn dài hơn, dài dòng hơn có vẻ không hay, nhưng khi xử lý các vấn đề OR, đây thường là lựa chọn tốt nhất:

```
1 SELECT
2     PRODUCT.ProductID,
3     PRODUCT.Name
4 FROM Production.Product PRODUCT
5 INNER JOIN Sales.SalesOrderDetail DETAIL
6 ON PRODUCT.ProductID = DETAIL.ProductID
7 UNION
8 SELECT
9     PRODUCT.ProductID,
10    PRODUCT.Name
11 FROM Production.Product PRODUCT
12 INNER JOIN Sales.SalesOrderDetail DETAIL
13 ON PRODUCT.rowguid = DETAIL.rowguid
```

Chúng tôi sẽ lấy từng thành phần của OR và biến nó thành câu lệnh SELECT. UNION kết nối tập tin và loại bỏ các bản sao. Vì vậy, kết quả trả về rất đáng mong đợi, số lần đọc đã giảm từ 1,2 triệu xuống còn 750 và truy vấn được thực hiện trong chưa đầy một giây, thay vì trong 2 giây.

Wildcard String Searches

Tìm kiếm chuỗi một cách hiệu quả là một thách thức và có nhiều cách để tìm kiếm các chuỗi không hiệu quả hơn là hiệu quả. Đối với các chuỗi được tìm kiếm thường xuyên, chúng ta cần đảm bảo rằng: Các chỉ mục có mặt trên các cột được tìm kiếm; Nếu không, chúng ta có thể sử dụng Full-Text Indexes? Nếu không, chúng ta có thể sử dụng hashes, n-grams hoặc một số giải pháp?

Nếu không sử dụng các tính năng bổ sung hoặc cân nhắc về thiết kế, SQL Server sẽ không thể tìm kiếm tốt. Nghĩa là, nếu tôi muốn phát hiện sự hiện diện của một chuỗi ở bất kỳ vị trí nào trong một cột, việc lấy dữ liệu đó sẽ không hiệu quả.

Ví dụ, trong tìm kiếm chuỗi bên dưới, chúng tôi đang kiểm tra LastName xem có bất kỳ sự xuất hiện nào của “For” ở bất kỳ vị trí nào trong chuỗi không.

```
1 SELECT
2     Person.BusinessEntityID,
3     Person.FirstName,
4     Person.LastName,
5     Person.MiddleName
6 FROM Person.Person
7 WHERE Person.LastName LIKE '%For%';
```

Kết quả truy vấn là quét Person.Person. Vì cách duy nhất để biết liệu một chuỗi con có tồn tại trong một cột văn bản hay không là lướt qua từng ký tự trong mỗi hàng, tìm kiếm sự xuất hiện của chuỗi đó. Trên một bảng nhỏ, điều này có thể được chấp nhận, nhưng đối với bất kỳ tập dữ liệu lớn nào, việc này sẽ chậm và khó chờ đợi.

N-Gram là các đoạn chuỗi có thể được lưu trữ riêng biệt với dữ liệu chúng ta đang tìm kiếm và có thể cung cấp khả năng tìm kiếm các chuỗi con mà không cần phải quét một bảng lớn. Ví dụ: hãy xem xét một hệ thống tìm kiếm trong đó độ dài tìm kiếm tối thiểu là 3 ký tự và từ được lưu trữ “Dinosaur”.

Dưới đây là các chuỗi con của Dinosaur có độ dài từ 3 ký tự trở lên (bỏ qua phần đầu của chuỗi): ino, inos, inosa, inosau, inosaur, nos, nosa, nosau, nosaur, osa, osau, osaur, sau, saur, aur.

```
1 SELECT
2     n_gram_table.my_big_table_id_column
3 FROM dbo.n_gram_table
4 WHERE n_gram_table.n_gram_data = 'Dino';
```

Giả sử n_gram_data được lập chỉ mục thì chúng tôi sẽ nhanh chóng trả về tất cả ID cho bảng lớn có từ Dino ở bất kỳ đâu trong đó. Bảng n-gram chỉ yêu cầu 2 cột và chúng ta có thể giới hạn kích thước của chuỗi n-gram bằng cách sử dụng các quy tắc ứng dụng được xác định ở trên. Ngay cả khi bảng này trở nên lớn, nó vẫn có khả năng cung cấp khả năng tìm kiếm rất nhanh.

Missing Indexes

Trong SQL Server, thông qua Management Studio GUI, Execution Plan XML, hoặc DVMs, chúng ta sẽ biết có các chỉ mục bị thiếu(Missing Index) từ đó có một cách khắc phục dễ dàng để cải thiện hiệu suất truy vấn.



Hình 2.14: Xem Missing Index

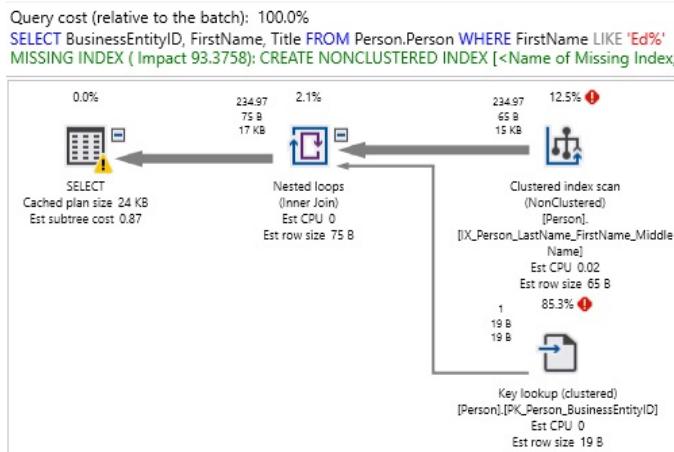
Dòng chữ màu xanh lá cây cung cấp cho chúng ta tất cả thông tin chi tiết về một New Index, nhưng chúng ta cần thực hiện một số công việc trước khi xem xét thực hiện lời khuyên của SQL Server như:

Có bất kỳ Index hiện có nào tương tự như Index này để có thể được sửa đổi phù hợp với Use-Case này không?

Tác động của Index cao đến mức nào? Nó sẽ cải thiện truy vấn lên 98% hay chỉ 5%.

Index này đã tồn tại nhưng vì lý do nào đó Query Optimizer không chọn nó?

Trong trường hợp trên, Impact 19% không quá ấn tượng. Chúng ta phải hỏi liệu truy vấn này có đủ quan trọng để đảm bảo sự cải thiện về hiệu suất hay không. Nếu nó được thực hiện một triệu lần một ngày thì có lẽ tất cả những nỗ lực cải thiện 19% này là đáng giá. Hãy xem xét một Index được đề xuất khác:



Hình 2.15: Xem Missing Index

Lần này, Index được đề xuất sẽ cải thiện 93% và xử lý cột không được lập Index (FirstName). Nếu đây hoàn toàn là một truy vấn được chạy thường xuyên thì việc thêm Index này có thể là một bước đi tốt. Vậy thì làm cách nào để biết khi nào hiệu suất của truy vấn là tối ưu?. Chúng ta cần đảm bảo không mắc phải bất kỳ lỗi nào dưới đây:

Over-Indexing: Khi một bảng có quá nhiều Index, các thao tác ghi sẽ trở nên chậm hơn vì mỗi UPDATE, DELETE và INSERT vào một cột được lập Index thì đều phải cập nhật các Index trên đó. Ngoài ra, các Index đó còn chiếm dung lượng bộ lưu trữ cũng như trong các bản sao lưu cơ sở dữ liệu.

Under-Indexing: Một bảng được lập Index quá ít sẽ không phục vụ các truy vấn một cách hiệu quả. Lý tưởng nhất là các truy vấn phổ biến nhất được thực hiện đối với một bảng sẽ được đánh Index. Các truy vấn ít thường xuyên hơn được đánh giá theo nhu cầu từng trường hợp và lập Index khi có lợi. Điều này xảy ra với các bảng có ít hoặc không có Non-Clustered Indexes, có thể là do lập ít Index. Trong những trường hợp này, Index để cải thiện hiệu suất khi cần.

No Clustered Index/Primary Key: Tất cả các bảng phải có Clustered Index/Primary Key. Clustered Index hầu như sẽ luôn hoạt động tốt hơn so với Heap và sẽ cung cấp cơ sở hạ tầng cần thiết để thêm Non-Clustered Indexes một cách hiệu quả khi cần. Primary Key cung cấp thông tin có giá trị cho Query Optimizer, giúp đưa ra quyết định thông minh khi tạo Execution Plans.

Query Hints

Gợi ý truy vấn(Query Hints) là các hướng dẫn, chỉ thị từ chúng ta đối với Query Optimizer để buộc nó hoạt động theo cách mà thông thường không làm. Query Hints thường được sử dụng khi gặp vấn đề về hiệu suất và việc thêm Query Hints sẽ khắc phục vấn đề một cách nhanh chóng. Có khá nhiều Hints có sẵn trong SQL Server.

Nguyên tắc chung là áp dụng Query Hint càng ít càng tốt, chỉ sau khi đã tiến hành nghiên cứu đầy đủ và chắc chắn rằng thay đổi sẽ không có tác động xấu. Chúng chỉ nên được sử dụng khi là phương án cuối cùng, tức là tất cả các lựa chọn khác đều thất bại. Một số Hint thường dùng như Nolock, Recompile, Merge/Hash/Loop và OptimizeFor.

Large Write Operations

Một thành phần của việc tối ưu hóa là sự tranh chấp. Khi thực hiện bất kỳ hành động nào với dữ liệu, các khóa sẽ được thực hiện đối với một lượng dữ liệu nhất định để đảm bảo rằng kết quả nhất quán và không ảnh hưởng đến các truy vấn khác. Tuy nhiên, việc tranh chấp liên tục trong một thời gian dài thì các truy vấn quan trọng có thể bị buộc phải chờ, dẫn đến việc người dùng không hài lòng về độ trễ.



Các thao tác ghi lớn là nguyên nhân dẫn đến sự tranh chấp vì chúng thường khóa toàn bộ bảng trong thời gian cần thiết để cập nhật dữ liệu, kiểm tra các ràng buộc, cập nhật chỉ mục và kích hoạt quy trình(nếu có). Trên một bảng không có trình kích hoạt hoặc khóa ngoại, lớn có thể là 50.000, 100.000 hoặc 1.000.000 hàng. Trên một bảng có nhiều ràng buộc và kích hoạt, số lớn có thể là 2.000.

2.4 Transaction processing

2.4.1 Giới thiệu

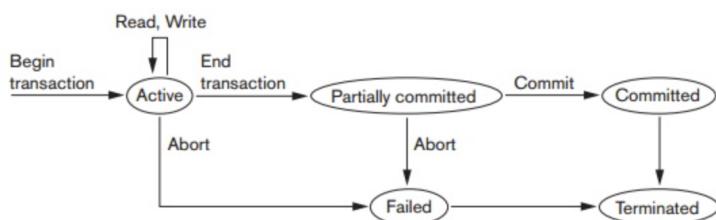
2.4.1.1 Định nghĩa

Giao tác(Transaction) là một đơn vị xử lý mang tính luận lý, gồm một hoặc nhiều thao tác trên Cơ sở dữ liệu. Đơn vị được nhắc đến là không được chia nhỏ và phải trọn vẹn. Một Transaction sẽ có điểm bắt đầu và điểm kết thúc, khi Transaction được thực hiện thì phải thực hiện từ đầu đến cuối, không thì tất cả phải được loại bỏ nếu có vấn đề trong quá trình thực hiện Transaction.

2.4.1.2 Transaction States

Xem như Transaction là một đối tượng có trạng thái gồm 5 trạng thái cơ bản.

- Active state
- Partially committed state
- Committed state
- Failed state
- Terminated State



Hình 2.16: Transaction states

Active state: Một Transaction đi vào trạng thái hoạt động ngay sau khi nó bắt đầu thực thi, bao gồm việc thực hiện các hoạt động READ và WRITE dữ liệu.

Partially committed: Khi Transaction kết thúc, một số giao thức Recovery cần đảm bảo rằng lỗi hệ thống sẽ không dẫn đến việc không thể ghi lại các thay đổi của Transaction (thường bằng cách ghi lại các thay đổi trong System Log).

Committed: Khi quá trình kiểm tra này thành công, Transaction được cho là đã đạt đến điểm Commit. Khi một Transaction được Commit nghĩa là nó đã kết thúc việc thực hiện thành công và tất cả các thay đổi của nó phải được ghi lại vĩnh viễn trong Cơ sở dữ liệu.

Failed state: Là những gì Transaction tác động lên Cơ sở dữ liệu nếu không thành công thì cần phải được tháo gỡ. Sau đó Transaction cần được khôi phục để hoàn tất các hoạt động của nó lên Cơ sở dữ liệu

Terminated: Transaction rời khỏi hệ thống. Thông tin Transaction sẽ được duy trì trong System Table khi Transaction đang chạy sẽ bị xóa khi kết thúc, chính thức ngừng hoàn toàn.

2.4.1.3 Phân loại Transaction trong SQL-Server

SQL-Server hoạt động trong 3 loại chính:

Autocommit Transaction: Mỗi câu lệnh riêng lẻ được coi là một Transaction.



Explicit Transaction: Là một Transaction mà chúng ta phải định nghĩa bắt đầu một Transaction(Begin transaction) và kết thúc một Transaction(Commit Transaction).

Implicit Transaction: Tự động khởi tạo ngay khi ở chế độ ON và SQL Server chạy, chỉ cần điều khiển Commit hay Rollback Transaction này. SQL-Server tự động bắt đầu một Transaction khi nó thực thi bất kỳ các lệnh sau: Alter Table, Creat, Delete, Drop, Fetch, Grant, Insert, Open, Revoke, Select, Truncate Table, Update.

2.4.1.4 Đặc điểm của Transaction

Atomicity: Toàn bộ các thao tác trong Transaction được thực hiện thành công. Nếu không thành công, tất cả các thao tác sẽ bị hủy tại điểm gặp lỗi và tất cả các thao tác trước đó sẽ được Rollback.

Consistency: Đặc tính này đảm bảo rằng toàn bộ dữ liệu sẽ được nhất quán sau khi một Transaction hoàn tất theo các quy tắc, ràng buộc, liên kết và Trigger đã được xác định.

Isolation: Tất cả các Transaction đều được cô lập khỏi các Transaction khác.

Durability: Các thay đổi của các Transaction đã Commit sẽ trở nên vĩnh viễn trong cơ sở dữ liệu.

2.4.2 Hiện thực hóa Transaction

Trong phần này nhóm sẽ thao tác trên Cơ sở dữ liệu đã được thiết lập cho ứng dụng Thương mại điện tử Le Livre(Chương 4) để hiện thực hóa các Transaction liên quan.

2.4.2.1 Autocommit

Mỗi câu lệnh đơn được thực hiện và tự động Commit ngay sau khi nó hoàn thành. Ở chế độ này, người ta không cần phải viết bất kỳ câu lệnh cụ thể nào để bắt đầu và kết thúc các Transaction. Mỗi câu lệnh được xem là 1 Transaction, đây là chế độ mặc định cho SQL Server Database Engine.

Ví dụ: Khi thực hiện câu lệnh INSERT sau đây, dữ liệu sẽ tự động được commit ngay sau khi câu lệnh hoàn thành, vì không có Transaction được bắt đầu tường minh.

```
1 INSERT INTO BOOK (Book_name, O_Price, Discount, Publish_year, Description, Ratings, Thumbnail, Reviews_N, Quantity)
2 VALUES
3 ('To Kill a Mockingbird', 200000, 0, 1960, 'A classic novel by Harper Lee', 5, 'mockingbird.jpg', 500, 50);
```

	Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description	Ratings	Thumbnail
1	1	To Kill a Mockingbird	200000	0	200000	1960	A classic novel by Harper Lee	5	mockingbird.jpg

Hình 2.17: Kết quả dùng Autocommit

Nhưng để quản lý giao dịch một cách rõ ràng hơn, có thể sử dụng các lệnh BEGIN TRANSACTION, COMMIT hoặc ROLLBACK để kiểm soát việc áp dụng hoặc hủy bỏ các thay đổi vào cơ sở dữ liệu.

2.4.2.2 Implicit Transaction

IMPLICIT_TRANSACTIONS là OFF

Khi chúng ta cài IMPLICIT_TRANSACTIONS là OFF thì các câu lệnh sẽ thực hiện các Transaction ở chế độ Autocommit. Nghĩa là, chúng bắt đầu và kết thúc Transaction một cách ngầm định. Vì vậy, nó giống như có một câu lệnh BEGIN TRANSACTION vô hình và một câu lệnh COMMIT TRANSACTION vô hình. Chúng ta không cần phải làm gì để Commit hoặc Rollback Transaction.



Ví dụ: chúng ta đặt IMPLICIT_TRANSACTIONS thành OFF và chạy câu lệnh SELECT. Điều này có nghĩa là câu lệnh SELECT chạy ở chế độ Autocommit. Do đó, Transaction được bắt đầu và kết thúc một cách ngầm định.

```
1 SELECT @@TRANCOUNT AS TransactionCount;
2 SET IMPLICIT_TRANSACTIONS OFF;
3 SELECT TOP 1 Book_name, Price FROM BOOK;
4 SELECT @@TRANCOUNT AS TransactionCount;
```

Hàm @@TRANCOUNT trả về số lượng câu lệnh BEGIN TRANSACTION trong phiên hiện tại và chúng ta có thể sử dụng hàm này để đếm số lượng Transaction cục bộ đang mở.

Results		Messages	
		TransactionCount	
1		0	

	Book_name	Price
1	To Kill a Mockingbird	200000

Results		Messages	
		TransactionCount	
1		0	

Hình 2.18: Kết quả Implicit Transaction khi OFF

IMPLICIT_TRANSACTIONS là ON

Khi chúng ta cài IMPLICIT_TRANSACTIONS là ON thì các câu lệnh sẽ hoạt động hơi khác một chút. Nó sẽ nhận được câu lệnh BEGIN TRANSACTION ẩn nhưng chúng không nhận được câu lệnh COMMIT TRANSACTION tương ứng. Điều này có nghĩa là chúng ta cần phải tự commit hoặc Rollback Transaction một cách rõ ràng trong câu truy vấn.

```
1 SELECT @@TRANCOUNT AS TransactionCount;
2 SET IMPLICIT_TRANSACTIONS ON;
3 SELECT TOP 1 Book_name, Price FROM BOOK;
4 SELECT @@TRANCOUNT AS TransactionCount;
```



Results		Messages
TransactionCount		
1	0	

	Book_name	Price
1	To Kill a Mockingbird	200000

Results		Messages
TransactionCount		
1	1	

Hình 2.19: Kết quả Implicit Transaction khi ON nhưng chưa Commit

@@TRANCOUNT trả về giá trị 1, có nghĩa là Transaction vẫn đang được tiến hành. @@TRANCOUNT trả về số lượng câu lệnh BEGIN TRANSACTION đã xảy ra trên kết nối hiện tại. Vì vậy, chúng ta thực sự cần phải Commit Transaction này (hoặc Rollback) để giảm @@TRANCOUNT xuống 0.

```
1 SELECT @@TRANCOUNT AS TransactionCount;
2 SET IMPLICIT_TRANSACTIONS ON;
3 SELECT TOP 1 Book_name, Price FROM BOOK;
4 COMMIT TRANSACTION;
5 SELECT @@TRANCOUNT AS TransactionCount;
```

Results		Messages
TransactionCount		
1	1	

	Book_name	Price
1	To Kill a Mockingbird	200000

Results		Messages
TransactionCount		
1	0	

Hình 2.20: Kết quả Implicit Transaction khi ON và Commit

2.4.2.3 Explicit Transaction

Để xác định một Explicit Transaction, chúng ta sẽ sử dụng lệnh BEGIN TRANSACTION và kết thúc bằng câu lệnh COMMIT hoặc ROLLBACK.

Sau khi xác định một Explicit Transaction thông qua lệnh BEGIN TRANSACTION, các tài nguyên liên quan sẽ nhận được khóa tùy thuộc vào mức độ cô lập của giao dịch. Vì lý do này, việc sử dụng giao

dịch ngắn nhất có thể sẽ giúp giảm thiểu các vấn đề về khóa. Câu lệnh sau bắt đầu một giao dịch và sau đó nó sẽ thay đổi năm xuất bản(Publish_year) cụ thể trong bảng BOOK.

```
1 BEGIN TRANSACTION UpdatePublishYear
2
3 DECLARE @BookID INT = 4
4 DECLARE @NewPublishYear INT = 2023
5 UPDATE BOOK
6 SET Publish_year = @NewPublishYear
7 WHERE Book_ID = @BookID
8
9 SELECT @@TRANCOUNT AS OpenTransactions
```

Results		Messages
		OpenTransactions
1		1

Hình 2.21: Kết quả dùng Explicit Transaction cập nhật năm xuất bản cuốn sách

Như chúng ta đã thực hiện trong phần trước, câu lệnh COMMIT TRAN sẽ áp dụng các thay đổi dữ liệu cho cơ sở dữ liệu và dữ liệu đã thay đổi sẽ trở thành vĩnh viễn. Bây giờ hãy hoàn thành giao dịch đang mở bằng câu lệnh COMMIT TRAN.

```
1 BEGIN TRANSACTION UpdatePublishYear
2
3 DECLARE @BookID INT = 4
4 DECLARE @NewPublishYear INT = 2023
5 UPDATE BOOK
6 SET Publish_year = @NewPublishYear
7 WHERE Book_ID = @BookID
8
9 SELECT @@TRANCOUNT AS OpenTransactions
10 COMMIT TRAN
11 SELECT @@TRANCOUNT AS OpenTransactions
```

Results		Messages
		TransactionCount
1		0

Hình 2.22: Kết quả khi thêm COMMIT TRAN

Mặt khác, chúng ta có thể dùng câu lệnh ROLLBACK TRANSACTION để giúp hoàn tác tất cả các sửa đổi dữ liệu được giao dịch áp dụng. Trong ví dụ sau, chúng ta sẽ thay đổi một hàng cụ thể nhưng việc sửa đổi dữ liệu này sẽ không tiếp tục.

```
1 BEGIN TRANSACTION UpdatePrice
2
3 DECLARE @BookID INT = 4
```



```
4 DECLARE @NewPrice INT = 110000
5 UPDATE BOOK
6 SET O_Price = @NewPrice
7 WHERE Book_ID = @BookID
8
9 SELECT * FROM BOOK WHERE @BookID=4
10
11 ROLLBACK TRAN
12
13 SELECT * FROM BOOK WHERE @BookID=4
```

Results Messages							
Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description	
1	To Kill a Mockingbird	200000	0	200000	1960	A classic novel by Harper Lee	
2	1984	250000	10	225000	1999	A dystopian novel by George Orwell	
3	The Great Gatsby	180000	5	171000	2005	A novel by F. Scott Fitzgerald	
4	Pride and Prejudice	110000	8	101200	2023	A novel by Jane Austen	
5	Moby Dick	300000	15	255000	2009	A novel by Herman Melville	
6	War and Peace	280000	0	280000	2015	A novel by Leo Tolstoy	
7	The Catcher in the Rye	240000	10	216000	2001	A novel by J.D. Salinger	
8	Pride and Prejudice	230000	8	211600	1932	A novel by Aldous Huxley	
9	Frankenstein	210000	0	210000	1998	A novel by Mary Shelley	
10	The Hobbit	260000	10	234000	2016	A fantasy novel by J.R.R. Tolkien	

Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description
1	To Kill a Mockingbird	200000	0	200000	1960	A classic novel by Harper Lee
2	1984	250000	10	225000	1999	A dystopian novel by George Orwell
3	The Great Gatsby	180000	5	171000	2005	A novel by F. Scott Fitzgerald
4	Pride and Prejudice	120000	8	102400	2023	A novel by Jane Austen
5	Moby Dick	300000	15	255000	2009	A novel by Herman Melville
6	War and Peace	280000	0	280000	2015	A novel by Leo Tolstoy
7	The Catcher in the Rye	240000	10	216000	2001	A novel by J.D. Salinger
8	Pride and Prejudice	230000	8	211600	1932	A novel by Aldous Huxley
9	Frankenstein	210000	0	210000	1998	A novel by Mary Shelley
10	The Hobbit	260000	10	234000	2016	A fantasy novel by J.R.R. Tolkien

Hình 2.23: Kết quả khi sử dụng ROLLBACK TRAN

Save Points trong Transactions

Điểm lưu trữ (Save Points) có thể được sử dụng để khôi phục bất kỳ phần cụ thể nào của giao dịch thay vì toàn bộ giao dịch. Vì vậy, chúng ta chỉ có thể khôi phục bất kỳ phần nào của giao dịch ở giữa sau điểm lưu và trước lệnh ROLLBACK. Để xác định Save Points trong giao dịch, chúng ta sử dụng cú pháp SAVE TRANSACTION. Ví dụ:

```
1 BEGIN TRANSACTION UpdatePrice
2
3 DECLARE @BookID INT = 4
4 DECLARE @NewPrice INT = 100000
5 UPDATE BOOK
6 SET O_Price = @NewPrice
7 WHERE Book_ID = @BookID
8
9 SAVE TRANSACTION InsertStatement1
10
11 DELETE BOOK WHERE Book_ID=1
12
13 SELECT * FROM BOOK
14 ROLLBACK TRANSACTION InsertStatement1
15 COMMIT
16 SELECT * FROM BOOK
```



Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description
1	1984	250000	10	225000	1999	A dystopia novel by George Orwell
2	The Great Gatsby	180000	5	171000	2005	A novel by F. Scott Fitzgerald
3	Pride and Prejudice	100000	8	92000	2023	A novel by Jane Austen
4	Moby Dick	300000	15	255000	2009	A novel by Herman Melville
5	War and Peace	280000	6	280000	2015	A novel by Leo Tolstoy
6	The Catcher in the Rye	240000	10	216000	2001	A novel by J.D. Salinger
7	Brave New World	230000	8	211600	1932	A novel by Aldous Huxley
8	Frankenstein	210000	0	210000	1998	A novel by Mary Shelley
9	The Hobbit	260000	10	234000	2016	A fantasy novel by J.R.R. Tolkien
Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description
1	To Kill a Mockingbird	200000	0	200000	1960	A classic novel by Harper Lee
2	1984	250000	10	225000	1999	A dystopian novel by George Orwell
3	The Great Gatsby	180000	5	171000	2005	A novel by F. Scott Fitzgerald
4	Pride and Prejudice	100000	8	92000	2023	A novel by Jane Austen
5	Moby Dick	300000	15	255000	2009	A novel by Herman Melville
6	War and Peace	280000	6	280000	2015	A novel by Leo Tolstoy
7	The Catcher in the Rye	240000	10	216000	2001	A novel by J.D. Salinger
8	Brave New World	230000	8	211600	1932	A novel by Aldous Huxley
9	Frankenstein	210000	0	210000	1998	A novel by Mary Shelley
10	The Hobbit	260000	10	234000	2016	A fantasy novel by J.R.R. Tolkien

Hình 2.24: Kết quả khi sử dụng Save Point

2.5 Concurrency control

Công cụ cơ sở dữ liệu SQL Server sử dụng các cơ chế sau để đảm bảo tính toàn vẹn của các giao dịch và duy trì tính nhất quán của cơ sở dữ liệu khi nhiều người dùng đang truy cập dữ liệu cùng một lúc:

Khóa Mỗi giao dịch yêu cầu khóa các loại khác nhau trên các tài nguyên, chẳng hạn như hàng, trang hoặc bảng, mà giao dịch phụ thuộc vào. Các ô khóa chặn các giao dịch khác sửa đổi tài nguyên theo cách có thể gây ra sự cố cho giao dịch yêu cầu khóa. Mỗi giao dịch giải phóng ô khóa của nó khi nó không còn phụ thuộc vào các tài nguyên bị khóa.

Lập phiên bản hàng Khi mức cách ly dựa trên phiên bản hàng được bật, công cụ cơ sở dữ liệu SQL Server duy trì phiên bản của mỗi hàng được sửa đổi. Các ứng dụng có thể chỉ định rằng một giao dịch sử dụng các phiên bản hàng để xem dữ liệu như nó tồn tại khi bắt đầu giao dịch hoặc truy vấn thay vì bảo vệ tất cả các lần đọc bằng khóa. Bằng cách sử dụng lập phiên bản hàng, khả năng thao tác đọc sẽ chặn các giao dịch khác sẽ giảm đáng kể.

Khóa và lập phiên bản hàng ngăn người dùng đọc dữ liệu không cam kết và ngăn nhiều người dùng cố gắng thay đổi cùng một dữ liệu cùng một lúc. Nếu không khóa hoặc lập phiên bản hàng, các truy vấn được thực thi đối với dữ liệu đó có thể tạo ra kết quả không mong muốn bằng cách trả về dữ liệu chưa được cam kết trong cơ sở dữ liệu.

Các ứng dụng có thể chọn các cấp độ cách ly giao dịch, xác định mức độ bảo vệ cho giao dịch khỏi các sửa đổi được thực hiện bởi các giao dịch khác. Gói ý cấp bảng có thể được chỉ định cho các câu lệnh Transact-SQL riêng lẻ để điều chỉnh thêm hành vi phù hợp với yêu cầu của ứng dụng.

2.5.1 Các mức cõ lập

SQL Server hỗ trợ nhiều loại điều khiển đồng thời. Người dùng chỉ định loại kiểm soát đồng thời bằng cách chọn mức cách ly giao dịch cho các kết nối hoặc tùy chọn đồng thời trên con trỏ.

Trong số năm mức cõ lập này, Đọc không cam kết, Đọc cam kết, Đọc lặp lại và Có thể tuân tự hóa đều thuộc mô hình đồng thời bi quan. Ảnh chụp nhanh được đưa ra theo mô hình tương tranh lạc quan.

Read uncommitted

Mức cách ly thấp nhất trong đó các giao dịch chỉ được cách ly đủ để đảm bảo rằng dữ liệu bị sai khác về mặt vật lý không được đọc. Ở cấp độ này, việc đọc sai, đọc ma, đọc không lặp lại được cho phép, vì vậy một giao dịch có thể thấy những thay đổi chưa được cam kết được thực hiện bởi các giao dịch khác.

Ví dụ, thực hiện giao dịch update và select dữ liệu cùng lúc.

Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description	Ratings	Thumbnail	Reviews_N	Quantity	Deleted
1	To Kill a Mockingbird	200000	0	200000	1960	A classic novel by Harper Lee	5	mockingbird.jpg	500	50	0
2	1984	250000	10	225000	1999	A dystopian novel by George Orwell	4	1984.jpg	300	50	0
3	The Great Gatsby	180000	5	171000	2005	A novel by F. Scott Fitzgerald	5	gatsby.jpg	450	90	0
4	Pride and Prejudice	220000	8	202400	2019	A novel by Jane Austen	5	pride.jpg	600	12	0
5	Moby Dick	300000	15	255000	2009	A novel by Herman Melville	4	mobydick.jpg	250	500	0
6	War and Peace	280000	0	280000	2015	A novel by Leo Tolstoy	5	warandpeace.jpg	350	80	0
7	The Catcher in the Rye	240000	10	216000	2001	A novel by J.D. Salinger	4	catcher.jpg	400	50	0
8	Brave New World	230000	8	211600	1932	A novel by Aldous Huxley	4	bravenewworld.jpg	280	60	0
9	Frankenstein	210000	0	210000	1998	A novel by Mary Shelley	4	frankenstein.jpg	320	70	0
10	The Hobbit	260000	10	234000	2016	A fantasy novel by J.R.R. Tolkien	5	hobbit.jpg	550	10	0

Hình 2.25: Dữ liệu bảng BOOK ban đầu

Update tên 1 cuốn sách, sau 10s thì rollback.

```
1 GO
2 BEGIN TRANSACTION;
3 UPDATE BOOK SET Book_name = 'Read Uncommitted' WHERE Book_ID=2;
4 WAITFOR DELAY '00:00:10';
5 ROLLBACK;
```

Dòng thời Select item sách tương ứng.

```
1 SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
2 SELECT * FROM BOOK WHERE Book_ID=2;
```

Kết quả sau khi chạy ở giao dịch này, tên sách đã thay đổi:

Results												
	Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description	Ratings	Thumbnail	Reviews_N	Quantity	Deleted
1	2	Read Uncommited	250000	10	225000	1999	A dystopian novel by George Orwell	4	1984.jpg	300	50	0

Hình 2.26: Lỗi đọc sai dữ liệu

Giao dịch với lệnh Select, dữ liệu tên sách được thay đổi bởi giao dịch Update nhưng chưa committed, khi đó hiển thị kết quả ngay cho người dùng mà không đợi nó committed. Vì vậy, khi giao dịch Update rollback, giao dịch select đã committed và không thể thay đổi kết quả, dẫn đến kết quả sai.

Read committed

Giao dịch chỉ thấy dữ liệu đã được cam kết trước khi truy vấn bắt đầu và không bao giờ thấy dữ liệu không được cam kết hoặc các thay đổi đã được thực hiện trong quá trình thực hiện truy vấn bằng các giao dịch đồng thời. SQL Server giữ các khóa ghi (thu được trên dữ liệu đã chọn) cho đến khi kết thúc giao dịch, nhưng các khóa đọc sẽ được giải phóng ngay khi thao tác SELECT được thực hiện xong. Đây là cấp độ mặc định của Công cụ cơ sở dữ liệu SQL Server. Tuy nhiên việc read committed không áp dụng cho lệnh insert. Đọc ma, đọc không lặp lại có thể xảy ra.

Ví dụ, thực hiện giao dịch update và select dữ liệu cùng lúc như ví dụ trên.

Update tên 1 cuốn sách và commit.

```
1 GO
2 BEGIN TRANSACTION;
3 UPDATE BOOK SET Book_name = 'Read Uncommited' WHERE Book_ID=2;
4 WAITFOR DELAY '00:00:10';
5 COMMIT;
```

Dòng thời Select item sách tương ứng.

```
1 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2 SELECT * FROM BOOK WHERE Book_ID=2;
```

Kết quả khi chạy ở giao dịch này, tên sách thay đổi sau khi chờ giao dịch update hoàn tất (vì khóa ghi chưa được giải phóng).

Chạy lại lệnh update với Book_name = 'Read Committed', điều kiện id > 0. Thay lệnh select thành insert như sau:

```
1 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2 INSERT INTO BOOK (Book_name, O_Price, Discount, Publish_year, Description, Ratings, Thumbnail, Reviews_N,
3 Quantity)
4 VALUES ('New book', 200000, 0, 1960, 'New book', 5, 'mockingbird.jpg', 500, 50)
```

Chạy đồng thời 2 giao dịch. Giao dịch với lệnh Insert, thêm 1 hàng mới với tên sách 'New book' 1 cách nhanh chóng. Mặc dù giao dịch update đã đọc và chỉnh sửa tất cả bản ghi có id > 0 và Book_name = 'Read Committed' nhưng bản ghi mới (id=12) được cập nhật sau khi bảng BOOK được cập nhật và trước khi giao dịch update kết thúc. Do giao dịch update tại thời điểm đọc các bản ghi thì giao dịch insert chưa commit nên chỉ có 11 hàng được chỉnh sửa.

	Results	Messages										
Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description	Ratings	Thumbnail	Reviews_N	Quantity	Deleted	
1	1	Read Commit	200000	0	200000	1960	A classic novel by Harper Lee	5	mockingbird.jpg	500	50	0
2	2	Read Commit	250000	10	225000	1999	A dystopian novel by George Orwell	5	1984.jpg	300	50	0
3	3	Read Commit	180000	5	171000	2005	A novel by F. Scott Fitzgerald	5	gatsby.jpg	450	90	0
4	4	Read Commit	220000	8	202400	2019	A novel by Jane Austen	5	pride.jpg	600	12	0
5	5	Read Commit	300000	15	255000	2009	A novel by Herman Melville	4	mobydick.jpg	250	500	0
6	6	Read Commit	280000	0	280000	2015	A novel by Leo Tolstoy	5	warandpeace.jpg	350	80	0
7	7	Read Commit	240000	10	216000	2001	A novel by J.D. Salinger	4	catcher.jpg	400	50	0
8	8	Read Commit	230000	8	211600	1932	A novel by Aldous Huxley	4	bravenewworld.jpg	280	60	0
9	9	Read Commit	210000	0	210000	1998	A novel by Mary Shelley	4	frankenstein.jpg	320	70	0
10	10	Read Commit	260000	10	234000	2016	A fantasy novel by J.R.R. Tolkien	5	hobbit.jpg	550	10	0
11	11	Read Commit	200000	0	200000	1960	New book	5	mockingbird.jpg	500	50	0
12	12	New book	200000	0	200000	1960	New book	5	mockingbird.jpg	500	50	0

Hình 2.27: Lỗi đọc ma

Repeatable read

SQL Server giữ các khóa đọc và ghi có được trên dữ liệu đã chọn cho đến khi kết thúc giao dịch. Tuy nhiên, do khóa phạm vi không được quản lý nên đọc ma có thể xảy ra. Mức isolation này hoạt động như mức read committed 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, nhưng có thể chèn thêm cột mới. Đọc ma có thể xảy ra.

Serializable Cấp độ cao nhất nơi các giao dịch hoàn toàn tách biệt với nhau. SQL Server giữ các khóa đọc và ghi có được trên dữ liệu đã chọn, chúng sẽ được phát hành khi kết thúc giao dịch. Khóa phạm vi được thu thập khi thao tác SELECT sử dụng mệnh đề WHERE có phạm vi, đặc biệt là để tránh các lần đọc ảo. Mức isolation này hoạt động như mức Repeatable read nhưng giao dịch khác không thể chèn các hàng mới với các giá trị khóa sẽ nằm trong phạm vi khóa được đọc bởi bất kỳ câu lệnh nào trong giao dịch hiện tại cho đến khi giao dịch hiện tại hoàn tất, thậm chí không thể đọc dữ liệu không được cam kết.

Read Committed Snapshot (RCSI)

- Khi tùy chọn cơ sở dữ liệu READ_COMMITTED_SNAPSHOT được BẬT, cách ly READ COMMITTED sử dụng phiên bản hàng để cung cấp tính nhất quán khi đọc ở cấp câu lệnh. Hoạt động đọc chỉ yêu cầu khóa cấp bằng SCH-S và không cần khóa trang hoặc hàng. Nghĩa là, Công cụ cơ sở dữ liệu SQL Server sử dụng phiên bản hàng để trình bày mỗi câu lệnh với ảnh chụp nhanh nhất quán về mặt giao dịch của dữ liệu như nó tồn tại ở đầu câu lệnh. Khóa không được sử dụng để bảo vệ dữ liệu khỏi các cập nhật của các giao dịch khác.
- Khi READ_COMMITTED_SNAPSHOT được đặt TẮT, đây là cài đặt mặc định, cách ly READ COMMITTED sử dụng các khóa chia sẻ để ngăn các giao dịch khác sửa đổi các hàng trong khi giao dịch hiện tại đang chạy thao tác đọc. Các khóa chia sẻ cũng chặn câu lệnh đọc các hàng được sửa đổi bởi các giao dịch khác cho đến khi giao dịch khác được hoàn thành. Cả hai cách triển khai đều đáp ứng định nghĩa ISO về cách ly READ COMMITTED.

Snapshot

Mức cõ lập ảnh chụp nhanh sử dụng phiên bản hàng để cung cấp tính nhất quán khi đọc ở cấp độ giao dịch. Hoạt động đọc không bị khóa trang hoặc hàng; chỉ có được khóa bằng. Khi đọc các hàng được sửa đổi bởi một giao dịch khác, chúng sẽ truy xuất phiên bản của hàng tồn tại khi giao dịch bắt đầu. Bạn chỉ có thể sử dụng tính năng cách ly ảnh chụp nhanh đối với cơ sở dữ liệu khi tùy chọn ALLOW_SNAPSHOT_ISOLATION cơ sở dữ liệu được BẬT. Theo mặc định, tùy chọn này được đặt TẮT cho cơ sở dữ liệu người dùng.

2.5.2 Cơ chế khóa trong công cụ cơ sở dữ liệu

Trước khi một giao dịch có được sự phụ thuộc vào trạng thái hiện tại của một phần dữ liệu, chẳng hạn như bằng cách đọc hoặc sửa đổi dữ liệu, nó phải tự bảo vệ mình khỏi tác động của một giao dịch khác sửa đổi cùng một dữ liệu. Giao dịch thực hiện điều này bằng cách yêu cầu khóa phần dữ liệu. Khóa có các chế độ khác nhau, chẳng hạn như chia sẻ hoặc độc quyền. Chế độ khóa xác định mức độ phụ thuộc của giao dịch vào dữ liệu. Không có giao dịch nào có thể được cấp một khóa xung đột với chế độ khóa đã được cấp trên dữ liệu đó cho một giao dịch khác. Nếu giao dịch yêu cầu chế độ khóa xung đột với khóa

đã được cấp trên cùng một dữ liệu, phiên bản của công cụ cơ sở dữ liệu SQL Server sẽ tạm dừng giao dịch yêu cầu cho đến khi khóa đầu tiên được phát hành.

Khi một giao dịch sửa đổi một phần dữ liệu, nó sẽ giữ một số khóa nhất định bảo vệ sửa đổi cho đến khi kết thúc giao dịch. Giao dịch giữ các khóa có được trong bao lâu để bảo vệ các hoạt động đọc tùy thuộc vào cài đặt mức cách ly giao dịch và việc khóa được tối ưu hóa có được bật hay không.

Tất cả các khóa được giữ bởi một giao dịch được giải phóng khi giao dịch hoàn tất (đã committed hoặc roll back).

Các ứng dụng thường không yêu cầu khóa 1 cách trực tiếp. Khóa được quản lý nội bộ bởi một phần của SQL Server được gọi là trình quản lý khóa. Khi một phiên bản của công cụ cơ sở dữ liệu SQL Server xử lý câu lệnh Transact-SQL, bộ xử lý truy vấn SQL Server xác định tài nguyên nào sẽ được truy cập. Bộ xử lý truy vấn xác định loại khóa nào được yêu cầu để bảo vệ từng tài nguyên dựa trên loại truy cập và cài đặt mức cách ly giao dịch. Bộ xử lý truy vấn sau đó yêu cầu các khóa thích hợp từ trình quản lý khóa. Trình quản lý khóa cấp khóa nếu không có sự xung đột các khóa được giữ bởi các giao dịch khác.

SQL Server sử dụng chiến lược khóa động để xác định các khóa hiệu quả nhất về mặt chi phí. SQL Server tự động xác định khóa nào phù hợp nhất khi truy vấn được thực thi, dựa trên các đặc điểm của lược đồ và truy vấn.

2.5.2.1 Độ chi tiết và cấu trúc phân cấp khóa

Công cụ cơ sở dữ liệu SQL Server có khóa đa chi tiết cho phép các loại tài nguyên khác nhau bị khóa bởi một giao dịch. Để giảm thiểu chi phí khóa, SQL Server khóa tài nguyên tự động ở mức phù hợp với tác vụ. Khóa ở độ chi tiết nhỏ hơn, chẳng hạn như hàng, làm tăng tính đồng thời nhưng có chi phí cao hơn vì phải giữ nhiều khóa hơn nếu nhiều hàng bị khóa. Khóa ở độ chi tiết lớn hơn, chẳng hạn như bảng, rất tốn kém về mặt đồng thời vì khóa toàn bộ bảng hạn chế quyền truy cập vào bất kỳ phần nào của bảng bằng các giao dịch khác. Tuy nhiên, nó có chi phí thấp hơn vì ít khóa hơn đang được duy trì.

Bảng sau đây hiển thị các tài nguyên mà công cụ cơ sở dữ liệu SQL Server có thể khóa.

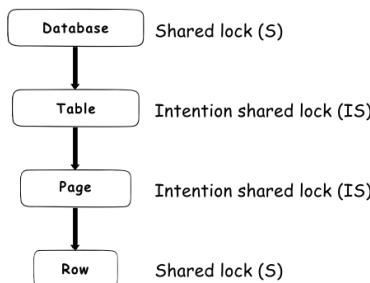
Resource	Description
RID	A row identifier used to lock a single row within a heap.
KEY	A row lock within an index used to protect key ranges in serializable transactions.
PAGE	An 8 kilobyte (KB) page in a database, such as data or index pages.
EXTENT	A contiguous group of eight pages, such as data or index pages.
HoBT ¹	A heap or B-tree. A lock protecting a B-tree (index) or the heap data pages in a table that doesn't have a clustered index.
TABLE ¹	The entire table, including all data and indexes.
FILE	A database file.
APPLICATION	An application-specified resource.
METADATA	Metadata locks.
ALLOCATION_UNIT	An allocation unit.
DATABASE	The entire database.
XACT ²	Transaction ID (TID) lock used in Optimized locking. See Transaction ID (TID) locking .

Hình 2.28: Các tài nguyên có thể khóa

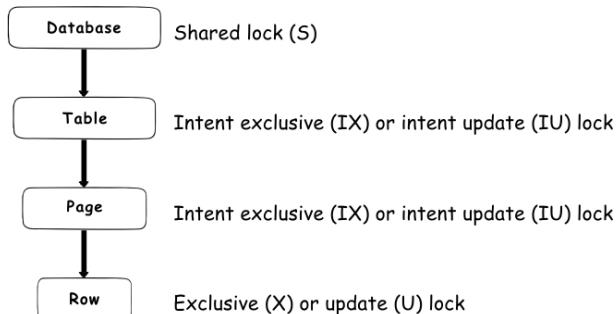
Công cụ cơ sở dữ liệu SQL Server thường phải có được khóa ở nhiều mức độ chi tiết để bảo vệ hoàn toàn

tài nguyên. Nhóm khóa này ở nhiều mức độ chi tiết được gọi là hệ thống phân cấp khóa.

Hệ thống phân cấp khóa bắt đầu với cơ sở dữ liệu ở cấp độ phân cấp cao nhất và đi xuống qua bảng và trang đến hàng ở cấp độ thấp nhất. Về cơ bản, luôn có một khóa chia sẻ ở cấp cơ sở dữ liệu được áp đặt bất cứ khi nào một giao dịch được kết nối với cơ sở dữ liệu. Khóa chia sẻ ở cấp cơ sở dữ liệu được áp dụng để ngăn chặn việc mất cơ sở dữ liệu hoặc khôi phục bản sao lưu cơ sở dữ liệu trên cơ sở dữ liệu đang sử dụng. Ví dụ: khi lệnh SELECT để đọc một số dữ liệu, khóa chia sẻ (S) sẽ được áp dụng ở cấp cơ sở dữ liệu, khóa chia sẻ ý định (IS) sẽ được áp dụng trên bảng và ở cấp độ trang, và khóa chia sẻ (S) trên chính hàng đó.



Trong trường hợp câu lệnh DML (tức là chèn, cập nhật, xóa), khóa chia sẻ (S) sẽ được áp dụng ở cấp cơ sở dữ liệu, khóa loại trừ dự định (IX) hoặc khóa cập nhật dự định (IU) sẽ được áp dụng trên bảng và trên cấp độ trang và khóa loại trừ hoặc cập nhật (X hoặc U) trên hàng.



2.5.2.2 Chế độ khóa

Shared locks (S):

- Khóa chia sẻ hay còn gọi là khóa đọc, được sử dụng cho thao tác đọc.
- Cho phép các giao dịch đồng thời đọc (SELECT) một tài nguyên nhưng không cho phép việc sửa đổi khi giao dịch giữ khóa chia sẻ đang diễn ra.
- Khóa chia sẻ (S) trên tài nguyên được phát hành ngay sau khi thao tác đọc hoàn tất.

Ví dụ, chạy lệnh select để đọc dữ liệu bảng BOOK, khóa chia sẻ được cấp cho giao dịch với session id 66.

```
Concurrency.sql - ...P-7UN412T\MSI (66)* | X
-- Shared Locks
SELECT * FROM dbo.BOOK;
```

Tiếp tục chạy lệnh select trên cùng vùng dữ liệu, khóa chia sẻ vẫn được cấp cho giao dịch với session id là 76.

```
SQLQuery1.sql - D...-7UN412T\MSI (76)* | X
USE DBMSAssignmentOrderBook;
SELECT * FROM dbo.BOOK WHERE Book_ID=2;
```

	request_session_id	request_mode	request_type	resource_type	resource_description
1	76	S	LOCK	DATABASE	
2	66	S	LOCK	DATABASE	
3	63	S	LOCK	DATABASE	

Update locks (U):

- Khóa U tương thích với khóa S, nhưng chỉ một giao dịch có thể giữ khóa U tại một thời điểm trên một tài nguyên nhất định. Nhiều giao dịch đồng thời có thể giữ khóa S, nhưng chỉ một giao dịch có thể giữ khóa U trên tài nguyên.
- Công cụ cơ sở dữ liệu đặt khóa cập nhật (U) khi nó chuẩn bị thực hiện cập nhật. Khóa cập nhật (U) cuối cùng được nâng cấp lên khóa loại trừ (X) để cập nhật hàng.
- Ngăn chặn một hình thức bế tắc phổ biến xảy ra khi nhiều phiên đang đọc, khóa và có khả năng cập nhật tài nguyên sau này.
- Được dùng bởi server để lọc ra các records được chỉnh sửa.
- Transaction giữ Update lock được phép ghi + đọc dữ liệu.

Exclusive locks (X):

- Khóa độc quyền (X) ngăn chặn quyền truy cập vào tài nguyên bằng các giao dịch đồng thời. Với khóa (X) độc quyền, không có giao dịch nào khác có thể sửa đổi dữ liệu; Các hoạt động đọc chỉ có thể diễn ra khi sử dụng gợi ý NOLOCK hoặc đọc mức cách ly không cam kết.
- Các câu lệnh sửa đổi dữ liệu, chẳng hạn như INSERT, UPDATE và DELETE kết hợp cả thao tác sửa đổi và đọc. Câu lệnh đầu tiên thực hiện các hoạt động đọc để thu thập dữ liệu trước khi thực hiện các hoạt động sửa đổi cần thiết. Do đó, các tuyên bố sửa đổi dữ liệu thường yêu cầu cả khóa chia sẻ và khóa loại trừ. Ví dụ: câu lệnh UPDATE có thể sửa đổi các hàng trong một bảng dựa trên phép nối với bảng khác. Trong trường hợp này, câu lệnh UPDATE yêu cầu khóa được chia sẻ trên các hàng được đọc trong bảng nối ngoài việc yêu cầu khóa độc quyền trên các hàng được cập nhật.

Ví dụ, thực thi lệnh cập nhật tên sách có id là 2, không kết thúc giao dịch bằng commit.

```
Concurrency.sql - ...P-7UN412T\MSI (60)*  SQLQuery3.sql - not connected*
-- Exclusive Locks
BEGIN TRAN
UPDATE BOOK SET Book_name='Test change' WHERE Book_ID=2;
```

Khóa loại trừ trên tài nguyên KEY được cấp cho giao dịch với session id là 60. Tiếp tục chạy lệnh select trên vùng dữ liệu, khóa chia sẻ được yêu cầu cho giao dịch với session id 66.

```
SQLQuery3.sql - DES...I (66) Executing...
USE DBMSAssignmentOrderBook;
SELECT * FROM dbo.BOOK;
```

	request_session_id	request_mode	request_type	resource_type	resource_description
1	60	S	LOCK	DATABASE	
2	66	S	LOCK	DATABASE	
3	63	S	LOCK	DATABASE	
4	60	IX	LOCK	PAGE	1:512
5	66	IS	LOCK	PAGE	1:512
6	60	X	LOCK	KEY	(61a06abd401c)
7	66	S	LOCK	KEY	(61a06abd401c)
8	60	IX	LOCK	OBJECT	
9	66	IS	LOCK	OBJECT	

Tuy nhiên, vì khóa loại trừ chưa giải phóng nên giao dịch này bị block bởi giao dịch với session id 60 và đợi khóa loại trừ giải phóng.

session_id	wait_duration_ms	wait_type	blocking_session_id	resource_description
10	50	CHECKPOINT_QUEUE	NULL	NULL
11	39	BROKER_TRANSMITTER	NULL	NULL
12	66	LCK_M_S	60	keylock hobtid=72057594045726720 dbid=5 id=lock1...

Lệnh select đọc dữ liệu có thể diễn ra khi cài đặt NOLOCK, ngay cả khi khóa loại trừ chưa giải phóng.

Concurrency.sql - ...P-7UN412T\MSI (64)*					SQLQuery2.sql - D...7UN412T\MSI (60)*					SQLQuery1.sql - D...7UN412T\MSI (61)*				
USE DBNameAssignmentOrderBook;					SELECT * FROM dbo.BOOK(nolock);									
100 %														
1	Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description	Ratings	Thumbnail	Reviews_N	Quantity	Deleted		
1	1	To Kill a Mockingbird	200000	0	200000	1960	A classic novel by Harper Lee	5	mockingbird.jpg	500	50	0		

Intent locks (I):

- SQL Server sử dụng khóa ý định để bảo vệ việc đặt khóa dùng chung (S) hoặc khóa loại trừ (X) trên tài nguyên thấp hơn trong hệ thống phân cấp khóa. Khóa ý định được mua trước khóa ở cấp thấp hơn và do đó, báo hiệu ý định đặt khóa ở mức thấp hơn.
- Khóa ý định bao gồm chia sẻ ý định (IS), loại trừ ý định (IX) và được chia sẻ với ý định loại trừ (SIX)... **Khóa chia sẻ với ý định loại trừ (SIX) được cấp khi thực thi cả câu lệnh select và update trong 1 giao dịch.**
- Mục đích:
 - Ngăn các giao dịch khác sửa đổi tài nguyên cấp cao hơn mà làm mất hiệu lực khóa ở cấp thấp hơn.
 - Cải thiện hiệu quả trong việc phát hiện xung đột khóa ở mức độ chi tiết cao hơn. Ví dụ, khi 1 giao dịch yêu cầu khóa X hoặc S trên bảng, nếu không có khóa ý định ở mức bảng, SQL Server sẽ quét khóa trên từng tài nguyên ở mức hàng để kiểm tra sự tương thích, xung đột khóa với khóa trên bảng. Ngược lại, nếu đặt khóa ý định X hoặc S trên bảng trước khi đặt ở từng hàng, khi 1 giao dịch yêu cầu khóa X hoặc S trên bảng/hàng, SQL Server có thể kiểm tra ngay với khóa hiện tại trên bảng.
- Ví dụ: khóa ý định chia sẻ (IS) được yêu cầu ở cấp bảng trước khi yêu cầu khóa dùng chung (S) trên các trang hoặc hàng trong bảng đó. Đặt khóa ý định ở cấp bảng sẽ ngăn một giao dịch khác sau đó có được khóa loại trừ trên bảng chứa trang đó.

Ví dụ, thực thi lệnh cập nhật tên sách có id là 2, không kết thúc giao dịch bằng commit.

Concurrency.sql - ...P-7UN412T\MSI (64)*		SQLQuery3.sql - not connected*			
-- Intent Locks					
BEGIN TRAN					
UPDATE BOOK SET Book_name='Test change' WHERE Book_ID=2;					

Khóa loại trừ trên tài nguyên KEY được cấp cho giao dịch với session id là 64. Trước khi yêu cầu khóa loại trừ trên KEY, khóa ý định loại trừ (IX) được đặt trên PAGE, báo hiệu việc đặt khóa ở cấp thấp hơn (trên KEY).

request_session_id	request_mode	request_type	resource_type	resource_description
1	S	LOCK	DATABASE	
2	S	LOCK	DATABASE	
3	64	IX	LOCK	PAGE
4	64	X	LOCK	KEY (61a06abd401c)
5	64	IX	LOCK	OBJECT

Schema locks:

- SQL Server sử dụng khóa sửa đổi lược đồ (Sch-M) trong câu lệnh (DDL), chẳng hạn như thêm cột hoặc drop bảng. Trong thời gian nó được giữ, khóa Sch-M ngăn chặn truy cập đồng thời vào bảng. Điều này có nghĩa là khóa Sch-M chặn tất cả các hoạt động bên ngoài cho đến khi khóa được giải phóng. Các câu lệnh (DDL), chẳng hạn như cắt bớt bảng, sử dụng khóa Sch-M để ngăn truy cập vào các bảng bởi các giao dịch đồng thời.

- SQL Server sử dụng khóa ẩn định lược đồ (Sch-S) khi biên dịch và thực thi truy vấn. Khóa Sch-S không chặn bất kỳ khóa giao dịch nào, bao gồm cả khóa độc quyền (X). Do đó, các giao dịch khác, bao gồm cả những giao dịch có khóa X trên bảng, tiếp tục chạy trong khi truy vấn đang được biên dịch. Tuy nhiên, các câu lệnh DDL đồng thời không thể được thực hiện trên bảng khi đang có khóa Sch-S trên tài nguyên.

Ví dụ, khóa Sch-M. Thực thi câu lệnh DDL drop cột Test_Schema trong bảng BOOK, không kết thúc giao dịch bằng commit. Khóa Sch-M được đặt trên tài nguyên OBJECT và METADATA cho giao dịch với session id là 64.

```
Concurrency.sql -...P-7UN412T\MSI (64)* SQLQuery3
ALTER TABLE BOOK ADD Test_Schema varchar(12)
/* SCH-M */
BEGIN TRAN
ALTER TABLE BOOK DROP COLUMN Test_Schema
```

	request_session_id	request_mode	request_type	resource_type	resource_description
1	64	S	LOCK	DATABASE	
2	63	S	LOCK	OBJECT	
3	64	IX	LOCK	OBJECT	
4	64	IX	LOCK	OBJECT	
5	64	IX	LOCK	OBJECT	
6	64	IX	LOCK	OBJECT	
7	64	Sch-M	LOCK	OBJECT	
8	64	Sch-M	LOCK	OBJECT	
9	64	X	LOCK	KEY	(aa19763ec7b8)
10	64	Sch-M	LOCK	OBJECT	
11	64	X	LOCK	KEY	(ea9500d50e2f)
12	64	Sch-M	LOCK	OBJECT	
13	64	Sch-M	LOCK	METADATA	class = 1, major_id = 901578250
14	64	Sch-M	LOCK	OBJECT	
15	64	Sch-M	LOCK	OBJECT	
16	64	Sch-M	LOCK	OBJECT	
17	64	X	LOCK	KEY	(021d50347b77)
18	64	Sch-M	LOCK	OBJECT	
19	64	Sch-M	LOCK	OBJECT	
20	64	X	LOCK	KEY	(862d4df3dd9f)
21	64	Sch-M	LOCK	OBJECT	
22	64	Sch-M	LOCK	OBJECT	

Tiếp tục thực thi giao dịch có lệnh select bảng BOOK trong session 58. Khóa Sch-M sẽ ngăn chặn các truy cập vào bảng, giao dịch này bị block bởi giao dịch với session id 64 và đợi khóa Sch-M giải phóng.

	session_id	wait_duration_ms	wait_type	blocking_session_id	resource_description
1	68	243004	LCK_M_SCH_S	64	objectlock lockPartition=0 objid=901578250 subre...

Ví dụ, khóa Sch-S. Thực thi câu lệnh DDL thêm cột Test_Schema trong bảng BOOK, không kết thúc giao dịch bằng commit. Khóa Sch-S được đặt trên tài nguyên METADATA, khóa Sch-M được đặt trên OBJECT để ngăn truy cập vào bảng khi thêm cột cho giao dịch với session id là 64.

```
Concurrency.sql -...P-7UN412T\MSI (64)* SQLQuery3.sql -
/* SCH-S */
BEGIN TRAN
ALTER TABLE BOOK ADD Test_Schema varchar(12)
```



	request_session_id	request_mode	request_type	resource_type	resource_description
1	64	IX	LOCK	OBJECT	
2	64	S	LOCK	DATABASE	
3	58	S	LOCK	DATABASE	
4	63	S	LOCK	DATABASE	
5	64	IX	LOCK	OBJECT	
6	64	S	LOCK	DATABASE	
7	64	IX	LOCK	OBJECT	
8	64	IX	LOCK	OBJECT	
9	64	Sch-S	LOCK	METADATA	data_space_id = 1
10	64	X	LOCK	KEY	(aa19763ec7b8)
11	64	X	LOCK	KEY	(0aa12e5aca93)
12	64	X	LOCK	KEY	(021d50347b77)
13	64	X	LOCK	KEY	(6619637c1923)
14	64	X	LOCK	KEY	(0482ebbe75f)
15	64	Sch-M	LOCK	OBJECT	

Bulk Update (BU):

- Khóa cập nhật hàng loạt (BU) cho phép nhiều luồng tải hàng loạt dữ liệu đồng thời vào cùng một bảng, đồng thời ngăn các quy trình truy cập bảng trong khi diễn ra. SQL Server sử dụng khóa Cập Nhật hàng loạt (BU) khi cả hai điều kiện sau là đúng.
 - Sử dụng câu lệnh Transact-SQL BULK INSERT hoặc hàm OPENROWSET(BULK) hoặc sử dụng một trong các lệnh Bulk Insert API.
 - Gợi ý TABLOCK được chỉ định hoặc tùy chọn khóa bảng trên bảng tải số lượng lớn.

Key-range locks:

- Khóa phạm vi khóa bảo vệ một loạt các hàng được bao gồm ngầm trong một tập hợp bản ghi được đọc bởi một câu lệnh Transact-SQL trong khi sử dụng mức cách ly giao dịch có thể tuân tự hóa. Khóa phạm vi ngăn chặn việc đọc ảo. Bằng cách bảo vệ phạm vi khóa giữa các hàng, nó cũng ngăn chặn việc chèn hoặc xóa ảo vào một tập hợp bản ghi được truy cập bởi một giao dịch.
- Một số loại khóa phạm vi: RangeS-S, RangeS-U, RangeX-X...

Ví dụ, thiết lập mức độ lặp khả tuân tự, thực hiện giao dịch select sách có tên 'Test 1'.

```
Concurrency.sql -...P-7UN412T\MSI (58)* -> X SQLQuery6.sql - D...-7
-- Key Range Locks
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
SELECT * FROM dbo.BOOK WHERE Book_name='Test 1';
```

Khóa RangeS-S được đặt trên tài nguyên KEY cho giao dịch với session id là 58.

	request_session_id	request_mode	request_type	request_status	resource_type	resource_description
1	58	S	LOCK	GRANT	DATABASE	
2	58	RangeS-S	LOCK	GRANT	KEY	()
3	58	IS	LOCK	GRANT	PAGE	1:514
4	58	IS	LOCK	GRANT	PAGE	1:1248
5	58	S	LOCK	GRANT	RID	1:1248:1
6	58	RangeS-S	LOCK	GRANT	KEY	(bb0048e930ef)
7	58	RangeS-S	LOCK	GRANT	KEY	(83257605c9a)
8	58	RangeS-S	LOCK	GRANT	KEY	(cb4a26fbe005)
9	58	RangeS-S	LOCK	GRANT	KEY	(5b346666453)
10	58	RangeS-S	LOCK	GRANT	KEY	(2b7e08742db9)
11	58	IS	LOCK	GRANT	OBJECT	
12	58	RangeS-S	LOCK	GRANT	KEY	(a2780df1115f)
13	58	RangeS-S	LOCK	GRANT	KEY	(d23263a3c8b5)
14	58	RangeS-S	LOCK	GRANT	KEY	(424c237ed5e3)
15	58	RangeS-S	LOCK	GRANT	KEY	(0a237ae5607c)
16	58	RangeS-S	LOCK	GRANT	KEY	(32064d6c0c09)

Tiếp theo, xóa hàng có tên sách 'Test 1', giao dịch không thể thực hiện được do khóa RangeS-S chưa giải phóng, bảo vệ các hàng trong phạm vi tên sách 'Test 1'.

```
SQLQuery1.sql - DES...I (56) Executing...* -> X Concurrency.sql -...
USE DBMSAssignmentOrderBook;
DELETE FROM dbo.BOOK WHERE Book_name='Test 1';
```

Results Messages						
session_id	wait_type	wait_duration_ms	blocking_session_id	resource_description		
7 31	PVS_PREALLOCATE	135335693	NULL			
8 56	LOCK_M_X	9633	58	keylock habid=72057594045726720 dbid=5 id=lock1...		

Giao dịch delete trong hàng đợi, do đó khóa U được cấp cho nó để sửa đổi dữ liệu chưa nâng cấp thành khóa X.

Results Messages						
request_session_id	request_mode	request_type	request_status	resource_type	resource_description	
11 56	IX	LOCK	GRANT	OBJECT		
12 56	IX	LOCK	GRANT	OBJECT		
13 58	RangeS-S	LOCK	GRANT	KEY	(bb0048e930ef)	
14 58	RangeS-S	LOCK	GRANT	KEY	(83257605c9a)	
15 58	RangeS-S	LOCK	GRANT	KEY	(cb4a26be905)	
16 58	RangeS-S	LOCK	GRANT	KEY	(5b346666453)	
17 58	RangeS-S	LOCK	GRANT	KEY	(2b7e08742db9)	
18 56	IX	LOCK	GRANT	OBJECT		
19 58	IS	LOCK	GRANT	OBJECT		
20 58	RangeS-S	LOCK	GRANT	KEY	(a2780df1115f)	
21 58	RangeS-S	LOCK	GRANT	KEY	(d23263e3c8b5)	
22 58	RangeS-S	LOCK	GRANT	KEY	(424c237ed5e3)	
23 58	RangeS-S	LOCK	GRANT	KEY	(0a237ae5607c)	
24 56	U	LOCK	GRANT	KEY	(32064d6c0c09)	
25 58	RangeS-S	LOCK	GRANT	KEY	(32064d6c0c09)	
26 56	X	LOCK	CONVERT	KEY	(32064d6c0c09)	

2.5.2.3 Tương thích khóa

Khả năng tương thích khóa kiểm soát xem nhiều giao dịch có thể có được khóa trên cùng một tài nguyên cùng một lúc hay không. Nếu tài nguyên đã bị khóa bởi một giao dịch khác, yêu cầu khóa mới chỉ có thể được chấp nhận nếu chế độ khóa được yêu cầu tương thích với chế độ khóa hiện có. Nếu chế độ khóa được yêu cầu không tương thích với khóa hiện có, giao dịch yêu cầu khóa mới sẽ đợi khóa hiện tại được giải phóng hoặc khoảng thời gian chờ khóa hết hạn.

Existing granted mode	I S	S	U	I X	S I X	X
Requested mode						
Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No	No
Update (U)	Yes	Yes	No	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No	No
Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
Exclusive (X)	No	No	No	No	No	No

Hình 2.29: Bảng tương thích khóa của các chế độ khóa thường gặp

Ví dụ, không có chế độ khóa nào tương thích với các khóa độc quyền. Trong khi khóa (X) độc quyền được giữ, không giao dịch nào khác có thể có được bất kỳ loại khóa nào (được chia sẻ, cập nhật hoặc độc quyền) trên tài nguyên đó cho đến khi khóa (X) độc quyền được giải phóng. Ngoài ra, nếu khóa (S) dùng chung đã được áp dụng cho một tài nguyên thì các giao dịch khác cũng có thể có được khóa dùng chung hoặc khóa cập nhật (U) trên mục đó ngay cả khi giao dịch đầu tiên chưa hoàn thành. Tuy nhiên, các giao dịch khác không thể có được khóa độc quyền cho đến khi khóa chung được giải phóng.

2.5.2.4 Chuyển đổi khóa

Chuyển đổi khóa là việc chuyển đổi loại khóa này sang loại khóa khác. Ví dụ, có thể có ba loại khóa chuyển đổi và được liệt kê bên dưới.

- Được chia sẻ với cập nhật ý định (SIU): là sự kết hợp giữa khóa chia sẻ (S) và khóa cập nhật ý định (IU). Một ví dụ điển hình của khóa này là khi một giao dịch đang sử dụng truy vấn được thực hiện bằng gọi ý và truy vấn PAGELOCK, sau đó là truy vấn cập nhật. Sau khi giao dịch có được khóa SIU trên bảng, truy vấn có gọi ý PAGELOCK sẽ có được khóa (S) được chia sẻ trong khi truy vấn cập nhật sẽ có được khóa cập nhật ý định (IU).
- Được chia sẻ với Ý định loại trừ (SIX): khóa này cho biết rằng giao dịch có ý định đọc tất cả tài nguyên ở hệ thống phân cấp thấp hơn và do đó có được khóa chia sẻ trên tất cả các tài nguyên có thứ bậc thấp hơn và do đó, sửa đổi một phần trong số đó, nhưng không phải tất cả. Khi làm như vậy, nó sẽ có được khóa ý định loại trừ (IX) đối với các tài nguyên phân cấp thấp hơn cần được sửa đổi. Trong thực tế, điều này có nghĩa là khi giao dịch có được khóa SIX trên bảng, nó sẽ có



được khóa ý định loại trừ (IX) trên các trang đã sửa đổi và khóa loại trừ (X) trên các hàng được sửa đổi.

- Cập nhật với Ý định loại trừ (UIX): Một giao dịch có khóa Cập nhật cũng có một số trang hoặc hàng bị khóa bằng khóa Độc quyền. Do đó, khóa UIX được yêu cầu ở mức bảng.

2.5.3 Deadlock

Deadlock xảy ra khi hai hoặc nhiều tác vụ chặn vĩnh viễn nhau bởi mỗi tác vụ có khóa tài nguyên mà các tác vụ khác đang cố gắng khóa.

Mô phỏng deadlock: Thực hiện 2 giao dịch ở 2 session khác nhau:

```
SQLQuery1.sql - D...-7UN412\MSI (56)*  X SQLQuery2.sql - D...-7UN412\MSI (79)*  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED  
BEGIN TRAN  
UPDATE AUTHOR SET Author_name='Test AUTHOR 2' WHERE Author_ID=2;  
WAITFOR DELAY '00:00:08'  
UPDATE BOOK SET Book_name='Test BOOK 2' WHERE Book_ID=1;  
  
Concurrency.sql - ...P-7UN412\MSI (58)*  X SQLQuery1.sql - D...-7UN412\MSI (56)*  
-- Deadlock  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED  
BEGIN TRAN  
UPDATE BOOK SET Book_name='Test BOOK 1' WHERE Book_ID=1;  
WAITFOR DELAY '00:00:08'  
UPDATE AUTHOR SET Author_name='Test AUTHOR 1' WHERE Author_ID=2;
```

Session 56 có khóa loại trừ trên bảng Author và yêu cầu 1 khóa trên Book, giao dịch này bị chặn cho đến khi giao dịch Session 58 kết thúc và giải phóng khóa nó có trên Book. Tương tự, Session 58 có khóa loại trừ trên Book và yêu cầu 1 khóa trên Author, giao dịch này bị chặn cho đến khi giao dịch Session 56 kết thúc và giải phóng khóa nó có trên Author. Cả 2 giao dịch chặn nhau và không thể thực thi.

Phát hiện và chấm dứt deadlock

SQL Server tự động phát hiện chu kỳ gián đoạn trong SQL Server. Công cụ cơ sở dữ liệu SQL Server chọn một trong các phiên làm nạn nhân bê tắc và giao dịch hiện tại bị chấm dứt với lỗi để phá vỡ bê tắc.

Phát hiện bê tắc được thực hiện bởi một luồng giám sát khóa định kỳ bắt đầu tìm kiếm thông qua tất cả các tác vụ trong SQL Server. Các điểm sau đây mô tả quá trình tìm kiếm:

- Khoảng thời gian mặc định là 5 giây.
- Nếu luồng giám sát khóa tìm bê tắc, khoảng thời gian phát hiện bê tắc sẽ giảm từ 5 giây xuống thấp nhất là 100 mili giây tùy thuộc vào tần suất bê tắc.
- Nếu luồng giám sát khóa ngừng tìm bê tắc, SQL Server tăng khoảng thời gian giữa các tìm kiếm lên 5 giây.
- Nếu bê tắc vừa được phát hiện, người ta cho rằng các luồng tiếp theo phải đợi khóa đang bước vào chu kỳ bê tắc. Một vài khóa đầu tiên chờ sau khi phát hiện bê tắc sẽ ngay lập tức kích hoạt tìm kiếm bê tắc thay vì đợi khoảng thời gian phát hiện bê tắc tiếp theo. Ví dụ: nếu khoảng thời gian hiện tại là 5 giây và bê tắc vừa được phát hiện, lần chờ khóa tiếp theo sẽ khởi động máy dò bê tắc ngay lập tức. Nếu khóa chờ này là một phần của bê tắc, nó sẽ được phát hiện ngay lập tức thay vì trong lần tìm kiếm bê tắc tiếp theo.

SQL Server thường chỉ thực hiện phát hiện gián đoạn định kỳ. Bởi số lượng deadlock gấp phải trong hệ thống thường nhỏ, việc phát hiện deadlock định kỳ giúp giảm chi phí phát hiện deadlock trong hệ thống.

Khi trình giám sát khóa bắt đầu tìm kiếm deadlock cho một luồng cụ thể, nó sẽ xác định tài nguyên mà luồng đang chờ đợi. Màn hình khóa sau đó tìm (các) chủ sở hữu cho tài nguyên cụ thể đó và đệ quy tiếp tục tìm kiếm bê tắc cho các luồng đó cho đến khi nó tìm thấy một chu kỳ. Một chu kỳ được xác định theo cách này tạo thành một bê tắc.

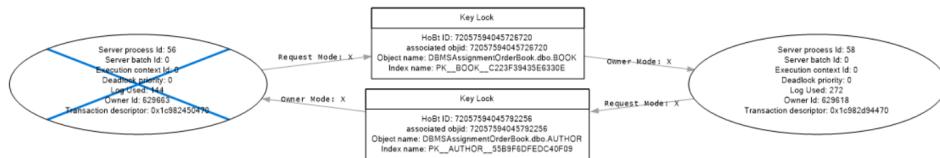
Sau khi phát hiện bế tắc, SQL Server kết thúc bế tắc bằng cách chọn một trong các luồng làm nạn nhân deadlock. SQL Server chấm dứt lô hiện tại đang được thực thi cho luồng, roll back giao dịch của nạn nhân bế tắc và trả về lỗi 1205 cho ứng dụng. Giao dịch cho nạn nhân bế tắc sẽ giải phóng tất cả các khóa do giao dịch nắm giữ. Điều này cho phép các giao dịch của các luồng khác được bỏ chặn và tiếp tục. Lỗi nạn nhân gián đoạn 1205 ghi lại thông tin về các chủ đề và tài nguyên liên quan đến gián đoạn trong nhật ký lỗi.

Kết quả thực thi các câu lệnh, thông báo lỗi tại session 56:

```
Msg 1205, Level 13, State 51, Line 5
Transaction (Process ID 66) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.
```

Giao dịch với process id 56 là nạn nhân của deadlock, roll back giao dịch này.

Đồ thị biểu diễn deadlock cho thấy khóa yêu cầu và khóa đang sở hữu của 2 giao dịch. Giao dịch có process id 58 sở hữu khóa X trên KEY của index trên trường khóa id bảng BOOK, yêu cầu khóa X trên KEY của index trên trường khóa id bảng AUTHOR. Giao dịch có process id 56 cũng tương tự. SQL Server chọn nạn nhân mà phiên chạy giao dịch ít tốn kém hơn để khôi phục. So với giao dịch 58, giao dịch 56 thực hiện tìm kiếm trên bảng Author với ít trường dữ liệu hơn và đó là lý do nó trở thành nạn nhân bế tắc.



Bởi vì bất kỳ ứng dụng nào gửi truy vấn Transact-SQL đều có thể được chọn làm nạn nhân bế tắc, các ứng dụng phải có trình xử lý lỗi có thể bẫy thông báo lỗi 1205. Nếu một ứng dụng không bẫy lỗi, ứng dụng có thể tiến hành mà không biết rằng giao dịch của nó đã được khôi phục và lỗi có thể xảy ra.

Triển khai trình xử lý lỗi bẫy thông báo lỗi 1205 cho phép ứng dụng xử lý tình huống bế tắc và thực hiện hành động khắc phục (ví dụ: tự động gửi lại truy vấn có liên quan đến bế tắc). Bằng cách gửi lại truy vấn tự động, người dùng không cần biết rằng đã xảy ra bế tắc. Ứng dụng nên tạm dừng một thời gian ngắn trước khi gửi lại truy vấn. Điều này mang lại cho giao dịch khác liên quan đến bế tắc có cơ hội hoàn thành và giải phóng các ổ khóa của nó tạo thành một phần của chu kỳ bế tắc. Điều này giảm thiểu khả năng bế tắc tái diễn khi truy vấn được gửi lại yêu cầu khóa của nó.

Theo mặc định, SQL Server chọn nạn nhân bế tắc mà phiên chạy giao dịch ít tốn kém nhất để khôi phục. Ngoài ra, người dùng có thể chỉ định mức độ ưu tiên của các phiên trong tình huống bế tắc bằng cách sử dụng câu lệnh. DEADLOCK_PRIORITY có thể được đặt thành THẤP, BÌNH THƯỜNG hoặc CAO hoặc cách khác có thể được đặt thành bất kỳ giá trị số nguyên nào trong phạm vi (-10 đến 10). Ưu tiên bế tắc mặc định là BÌNH THƯỜNG. Nếu hai phiên có mức độ ưu tiên bế tắc khác nhau, phiên có mức độ ưu tiên thấp hơn được chọn là nạn nhân bế tắc. Nếu cả hai phiên đều có cùng mức độ ưu tiên bế tắc, phiên có giao dịch ít tốn kém nhất để quay trở lại sẽ được chọn. Nếu các phiên liên quan đến chu kỳ bế tắc có cùng mức độ ưu tiên bế tắc và cùng chi phí, nạn nhân được chọn ngẫu nhiên.

Để giúp giảm thiểu bế tắc:

- Truy cập các đối tượng theo cùng một thứ tự.
- Giữ giao dịch ngắn và trong một đợt.
- Sử dụng mức cách ly thấp hơn.
- Sử dụng mức cách ly dựa trên lập phiên bản hàng.

2.5.4 Mức cách ly dựa trên phiên bản hàng trong SQL Server

Lập phiên bản hàng là cơ chế sao chép, tạo ra phiên bản hàng mới khi ghi khi một hàng được sửa đổi hoặc xóa. Điều này yêu cầu rằng trong khi giao dịch đang chạy, phiên bản cũ của hàng phải có sẵn cho

các giao dịch yêu cầu trạng thái nhất quán giao dịch trước đó. Lập phiên bản hàng được sử dụng để làm như sau:

- Bất kỳ hàng nào được sửa đổi bởi các trigger (sửa, xóa) đều được lập phiên bản. Điều này bao gồm các hàng được sửa đổi bởi câu lệnh trigger, cũng như bất kỳ sửa đổi dữ liệu nào được thực hiện bởi trigger.
- Hỗ trợ các hoạt động lập chỉ mục chỉ định tùy chọn ONLINE.
- Hỗ trợ các cấp độ cách ly giao dịch dựa trên phiên bản hàng. Một mức cách ly mới, READ COMMITTED, sử dụng lập phiên bản hàng để cung cấp tính nhất quán đọc ở cấp độ câu lệnh. Một cấp độ cách ly mới, SNAPSHOT, để cung cấp tính nhất quán đọc ở cấp độ giao dịch.

Sử dụng lập phiên bản hàng cho các giao dịch cam kết đọc và chụp ảnh nhanh gồm hai bước:

1. Đặt một hoặc cả hai tùy chọn database READ_COMMITTED_SNAPSHOT và ALLOW_SNAPSHOT_ISOLATION BẬT.
2. Đặt mức cách ly giao dịch thích hợp trong một ứng dụng:
 - Khi tùy chọn database READ_COMMITTED_SNAPSHOT được BẬT, các giao dịch đặt mức cách ly DỌC CAM KẾT sử dụng lập phiên bản hàng.
 - Khi tùy chọn database ALLOW_SNAPSHOT_ISOLATION được BẬT, các giao dịch có thể đặt mức cách ly ảnh chụp nhanh.

Khi các tùy chọn cơ sở dữ liệu trên BẬT, các bản sao logic (phiên bản) được duy trì cho tất cả các sửa đổi dữ liệu được thực hiện trong cơ sở dữ liệu. Mỗi khi một hàng được sửa đổi bởi một giao dịch cụ thể, phiên bản của công cụ cơ sở dữ liệu SQL Server lưu trữ một phiên bản của hình ảnh đã cam kết trước đó của hàng. Mỗi phiên bản được đánh dấu bằng số thứ tự giao dịch của giao dịch đã thực hiện thay đổi. Các phiên bản của các hàng đã sửa đổi được xâu chuỗi bằng cách sử dụng danh sách liên kết. Giá trị hàng mới nhất luôn được lưu trữ trong cơ sở dữ liệu hiện tại và được xâu chuỗi các phiên bản được lưu trữ trong tempdb.

Ví dụ, đặt tùy chọn READ_COMMITTED_SNAPSHOT. Đặt mức cách ly READ_COMMITTED cho 2 giao dịch đồng thời.

```
Concurrency.sql - ...P-7UN412T\MSI (58)* ALTER DATABASE DBMSAssignmentOrderBook SET READ_COMMITTED  
SELECT * FROM dbo.BOOK  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED  
BEGIN TRAN  
UPDATE BOOK SET Book_name='Row version' WHERE Book_ID=1;  
WAITFOR DELAY '00:00:08'  
COMMIT
```

```
SQLQuery5.sql - D...-7UN412T\MSI (60)* USE DBMSAssignmentOrderBook;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED  
BEGIN TRAN  
SELECT * from BOOK WHERE Book_ID=1  
COMMIT
```

Kết quả thực thi tại session 60, giao dịch không bị block và chờ giao dịch session 58, tên sách không thay đổi.

Book_ID	Book_name	O_Price	Discount	Price	Publish_year	Description	Ratings	Thumbnail	Reviews_N	Quantity	Deleted
1	Read Commit	200000	0	200000	1960	A classic novel by Harper Lee	5	Mockingbird.jpg	500	50	0

Với mức cõ lập READ_COMMITTED thông thường, giao dịch này phải chờ giao dịch session 58 thực thi và cho ra kết quả tên sách đã thay đổi. Nhưng 1 phiên bản mới của hàng đã cam kết trước đó được tạo ra cùng với số thứ tự giao dịch khi thực hiện chỉnh sửa dữ liệu tại session 58, giao dịch tại session 60 sẽ đọc phiên bản mới nhất, kết quả là tên sách không thay đổi.



Hành vi khi đọc dữ liệu:

Tất cả các truy vấn, bao gồm cả các giao dịch chạy ở mức cách ly dựa trên phiên bản hàng, đều có được khóa Sch-S (khóa lược đồ ổn định) trong quá trình biên dịch và thực thi. Vì điều này, các truy vấn bị chặn khi một giao dịch đồng thời giữ khóa Sch-M (sửa đổi lược đồ) trên table. Ví dụ: câu lệnh DDL thu được khóa Sch-M trước khi sửa đổi thông tin lược đồ của bảng. Các giao dịch truy vấn, bao gồm cả những giao dịch chạy ở mức cô lập dựa trên phiên bản hàng, đều bị chặn khi cố gắng lấy khóa Sch-S. Ngược lại, một truy vấn có khóa Sch-S sẽ chặn một giao dịch đồng thời cố gắng lấy khóa Sch-M.

Khi một giao dịch sử dụng mức cô lập ảnh chụp nhanh (SNAPSHOT) bắt đầu, SQL Server sẽ ghi lại tất cả các giao dịch hiện đang hoạt động. Khi giao dịch chụp nhanh đọc một hàng có chuỗi phiên bản, SQL Server sẽ theo chuỗi và truy xuất hàng có số thứ tự giao dịch là:

- Gần nhất nhưng thấp hơn số thứ tự của giao dịch chụp nhanh đang đọc hàng.
- Không có trong danh sách các giao dịch đang hoạt động khi giao dịch chụp nhanh bắt đầu.

Hoạt động đọc được thực hiện bởi giao dịch chụp nhanh truy xuất phiên bản cuối cùng của mỗi hàng đã được cam kết tại thời điểm giao dịch chụp nhanh bắt đầu. Điều này cung cấp một ảnh chụp nhanh nhất quán về mặt giao dịch của dữ liệu như nó đã tồn tại khi bắt đầu giao dịch.

Các giao dịch đã cam kết đọc (READ COMMITTED) sử dụng phiên bản hàng hoạt động theo cách tương tự. Sự khác biệt là giao dịch được cam kết đọc không sử dụng số thứ tự giao dịch của chính nó khi chọn phiên bản hàng. Mỗi khi một câu lệnh được bắt đầu, giao dịch đã cam kết đọc sẽ đọc số thứ tự giao dịch mới nhất được cấp cho phiên bản đó của Công cụ cơ sở dữ liệu SQL Server. Đây là số thứ tự giao dịch được sử dụng để chọn phiên bản hàng chính xác cho câu lệnh đó. Điều này cho phép các giao dịch đã cam kết đọc có thể xem ảnh chụp nhanh dữ liệu khi nó tồn tại ở đầu mỗi câu lệnh.

Hành vi khi chỉnh sửa dữ liệu:

Trong giao dịch được cam kết đọc bằng cách sử dụng phiên bản hàng, việc lựa chọn các hàng để cập nhật được thực hiện bằng cách quét trong đó khóa cập nhật (U) được lấy trên hàng dữ liệu khi giá trị dữ liệu được đọc. Điều này giống như giao dịch được cam kết đọc không sử dụng phiên bản hàng. Nếu hàng dữ liệu không đáp ứng tiêu chí cập nhật, khóa cập nhật sẽ được giải phóng trên hàng đó và hàng tiếp theo sẽ bị khóa và quét.

Các giao dịch chạy trong chế độ cách ly ảnh chụp nhanh có cách tiếp cận lạc quan đối với việc sửa đổi dữ liệu bằng cách lấy các khóa trên dữ liệu trước khi chỉ thực hiện sửa đổi để thực thi các ràng buộc. Nếu không, dữ liệu sẽ không được khóa cho đến khi dữ liệu được sửa đổi. Khi một hàng dữ liệu đáp ứng tiêu chí cập nhật, giao dịch chụp nhanh sẽ xác minh rằng hàng dữ liệu chưa bị sửa đổi bởi một giao dịch đồng thời đã cam kết sau khi giao dịch chụp nhanh bắt đầu. Nếu hàng dữ liệu đã được sửa đổi bên ngoài giao dịch chụp nhanh thì sẽ xảy ra xung đột cập nhật và giao dịch chụp nhanh sẽ bị chấm dứt. Xung đột cập nhật được xử lý bởi SQL Server và không có cách nào để tắt tính năng phát hiện xung đột cập nhật.

Giám sát lập phiên bản hàng và kho phiên bản

Để giám sát phiên bản hàng, lưu trữ phiên bản và quy trình cách ly ảnh chụp nhanh cho hiệu suất và sự cố, SQL Server cung cấp các công cụ dưới dạng Dynamic Management Views (DMVs) và bộ đếm hiệu suất trong Windows System Monitor.



2.6 Backup và recovery

2.6.1 Xây dựng giải pháp High Availability và Disaster Recovery

Bất kể một Database Engine nào cũng phải đảm bảo chạy ổn định và mang tính sẵn sàng cao cho người dùng, cũng như là việc đảm bảo được khả năng khôi phục hệ thống trong trường hợp có bất kỳ lỗi nào xảy ra, dữ liệu phải được mất mát tối thiểu, chấp nhận được. Đó cũng là mục đích chính của hai giải pháp này.

2.6.1.1 Những lý do dẫn đến máy chủ bị hỏng

Có những nguyên nhân dẫn đến máy chủ(Server) bị hỏng(Down), Database không truy cập được, làm mất mát dữ liệu như:

- Sự cố liên quan đến nguồn điện, dẫn đến sập nguồn.
- Truy cập quá nhiều dẫn đến hệ thống bị quá tải.
- Dung lượng ổ cứng bị đầy, tràn bộ nhớ.
- Lỗi do Networking.
- Lỗi phần mềm khi nâng cấp lên phiên bản mới(Upgrade/Patching).
- Lỗi do yếu tố con người(Human Error)
- Bị tấn công(Hacking).

Trong đó lỗi do phần mềm khi nâng cấp lên phiên bản mới, lỗi này sẽ có 2 dạng. Khi phần mềm do tổ chức, công ty đó tự phát triển, vì một lý do nào đó khiến lỗi về Code, có Bug làm cho Database không truy cập được. Thứ 2, bản thân Database Engine cũng là một phần mềm, cứ một khoảng thời gian thì các hằng sẽ tung ra bản vá lỗi, hay các bản nâng cấp bổ sung thêm tính năng, và khi chúng ta nâng cấp lên cũng sẽ có trường hợp khiến chúng ta không làm việc được vì thiếu thư viện, điều kiện nào đó mà phiên bản mới yêu cầu nhưng chúng ta không có, dẫn đến lỗi về Database. Ngoài ra, trong các lỗi trên thì lỗi do yếu tố con người là điều mà các công ty, doanh nghiệp không thể tránh khỏi. Với muôn vàn lý do, gọi chung là Human Error.

Vì vậy chúng ta cần có những giải pháp để hạn chế tối đa những lỗi trên xảy ra. Bản thân các hằng cũng phải phát triển ra các công cụ, những giải pháp để hạn chế lỗi này, đặc biệt là những hệ thống được xây dựng và phát triển trên Cloud.

2.6.1.2 Sự ảnh hưởng nếu không có giải pháp Database Backup

Riêng trong phạm vi của Database, khi xảy ra sự cố sẽ làm hệ thống mất dữ liệu, ảnh hưởng tới người dùng và nghiệp vụ của công ty/ tổ chức. Chúng ta cần xây dựng, chiến lược Database Backup - một công cụ giải pháp điển hình, luôn được tính đến đầu tiên trong bất kỳ chiến lược liên quan đến việc xây dựng các giải pháp về Database Disaster và Recovery.

2.6.2 Giải pháp Database Backup trong Microsoft SQL Server

Có nhiều phương pháp để chúng ta thực hiện Backup Database trong Microsoft SQL Server, trong đó có 3 phương pháp phổ biến nhất:

2.6.2.1 Full Backup

Full Backup thì ta sẽ Backup toàn bộ tất cả mọi thư của Database đó, bao gồm cả Data File và Log File. Phù hợp cho khi Backup không quá thường xuyên, hoặc Database của chúng ta tương đối nhỏ, không tồn quá nhiều về không lưu trữ.



2.6.2.2 Differential Backup

Differential Backup sẽ ghi nhận Backup lại những sự thay đổi nào từ lần Full Backup cuối cùng. Có nghĩa là để thực hiện Differential Backup thì trước tiên chúng ta cần có một Full Backup và sau đó thực hiện Differential Backup.

Ví dụ tối chủ nhật hàng tuần chúng ta thực hiện Full Backup thì qua ngày sáng thứ 2 khi thực hiện Differential Backup thì chúng ta chỉ Backup những giao dịch, dữ liệu nào được phát sinh từ tối chủ nhật đến thời điểm ta thực hiện Differential Backup sáng thứ 2.

2.6.2.3 Transaction Log Backup

Transaction Log Backup là việc Backup lại File Log của chúng ta. Trước khi dữ liệu được thực sự ghi vào Database thì trước tiên nó được ghi vào File Log(ghi nhận bất kỳ sự thay đổi về dữ liệu). Transaction Log Backup ứng dụng khi chúng ta mong muốn sẽ Backup hệ thống một cách thường xuyên(mỗi 15p phút, 30 phút...), thay vì sử dụng Differential Backup hay Full Backup làm tốn thời gian và không gian lưu trữ.

2.6.3 Xây dựng Backup Strategy

Khi chúng ta giải quyết bài toán liên quan đến Database Disaster thì chúng ta cần tìm hiểu 2 khái niệm trong quá trình xây dựng chiến lược Database Backup:

2.6.3.1 Recovery Point Objective(RPO)

Recovery Point Objective là chỉ số mô tả khoảng thời gian tối đa dữ liệu bị mất khi Disaster xảy ra. Ví dụ hệ thống cho phép tối đa mất dữ liệu trong vòng 15 phút, thì RPO là 15 phút(còn trước đó 15 phút thì dữ liệu vẫn phải có khả năng khôi phục). Với Backup thì chỉ số này quan trọng hơn.

2.6.3.2 Recovery Time Objective(RTO)

Recovery Time Objective là chỉ số cho phép hệ thống được Downtime tối đa trong bao lâu. Ví dụ khi hệ thống bị mất kết nối thì nó cho phép thời gian tối đa bị mất kết nối là 15 hay 20 phút để khôi phục lại.

2.6.3.3 Backup Strategy

Có các chiến lược Backup chúng ta có thể sử dụng như sau:

- Full Backup.
- Full + Differential Backup
- Full + Differential + Transaction Log Backup.

2.6.4 Thực hành Backup và Recovery

Trong phần này nhóm sẽ thao tác trên Cơ sở dữ liệu đã được thiết lập cho ứng dụng Thương mại điện tử Le Livre(Chương 4) để hiện thực hóa các Transaction liên quan.

2.6.4.1 Backup Full Database

```
1 BACKUP DATABASE [DBMSAssignmentOrderBook]
2 TO DISK = '/var/opt/mssql/data/DBMSAssignmentOrderBook.bak' WITH NOFORMAT, NOINIT,
3 NAME = 'DBMSAssignmentOrderBook-Full Database Backup', SKIP, NOREWIND, NOUNLOAD, COMPRESSION, STATS = 10
```

Kiểm tra File Backup để đảm bảo File Backup là Valid.



```
1 DECLARE @backupSetId as int
2 SELECT @backupSetId = position from msdb..backupset
3 where database_name=N'DBMSAssignmentOrderBook'
4 and backup_set_id=(select max(backup_set_id)
5 from msdb..backupset where database_name=N'DBMSAssignmentOrderBook' )
6 IF @backupSetId is null begin
7 raiserror(N'Verify failed. Backup information for database ''DBMSAssignmentOrderBook'' not found.', 16, 1)
8 END
9 RESTORE VERIFYONLY FROM DISK = N'/var/opt/mssql/data/DBMSAssignmentOrderBook.bak'
10 WITH FILE = @backupSetId, NOUNLOAD, NOREWIND
11 GO
```

Messages

10:28:44 PM Started executing query at Line 1111
The backup set on file 1 is valid.
Total execution time: 00:00:00.109

Hình 2.30: Kết quả kiểm tra File Full Backup

2.6.4.2 Insert một Record mới lần 1

```
1 INSERT INTO BOOK (Book_name, O_Price, Discount, Publish_year, Description, Ratings, Thumbnail, Reviews_N, Quantity)
2 VALUES
3 ('The Alchemist', 800000, 50, 2000, 'Helps you dare to live your dreams, without fear of failure',
4 5, 'new_alchemist.jpg', 50, 100);
```

11	11	The Alchemist	800000	50	400000	2000	Helps you dare to live your dreams, w:
----	----	---------------	--------	----	--------	------	--

Hình 2.31: Kết quả thêm một quyển sách mới lần 1

2.6.4.3 Thực hiện Differential Backup Database

```
1 BACKUP DATABASE [DBMSAssignmentOrderBook]
2 TO DISK = N'/var/opt/mssql/data/DBMSAssignmentOrderBook_Diff.DIF' WITH DIFFERENTIAL, NOINIT,
3 NAME = N'DBMSAssignmentOrderBook-Diff Database Backup', SKIP, NOREWIND, NOUNLOAD, COMPRESSION, STATS = 10
```

Kiểm tra File Backup để đảm bảo File Backup là Valid.

```
1 DECLARE @backupSetId as int
2 SELECT @backupSetId = position from msdb..backupset
3 where database_name=N'DBMSAssignmentOrderBook'
4 and backup_set_id=(select max(backup_set_id) from msdb..backupset
5 where database_name=N'DBMSAssignmentOrderBook' )
6 IF @backupSetId is null begin
7 raiserror(N'Verify failed. Backup information for database ''DBMSAssignmentOrderBook'' not found.', 16, 1)
8 END
9 RESTORE VERIFYONLY FROM DISK = N'/var/opt/mssql/data/DBMSAssignmentOrderBook_Diff.DIF'
10 WITH FILE = @backupSetId, NOUNLOAD, NOREWIND
11 GO
```



Messages

11:17:23 PM Started executing query at Line 1127
The backup set on file 1 is valid.
Total execution time: 00:00:00.049

Hình 2.32: Kết quả kiểm tra File Differential Backup

2.6.4.4 Insert một Record mới lần 2

```
1 INSERT INTO BOOK (Book_name, O_Price, Discount, Publish_year, Description, Ratings, Thumbnail, Reviews_N, Quantity)
2 VALUES
3 ('The Alchemist 2', 1000000, 50, 2000, 'Helps you dare to live your dreams, without fear of failure',
4 5, 'new_alchemist_2.jpg', 50, 100);
```

The screenshot shows a table with columns: ID, Price, Book Name, Publish Year, Rating, Reviews, and Description. A new row has been added with the following values: ID=12, Price=1000000, Book Name='The Alchemist 2', Publish Year=2000, Rating=50, Reviews=500000, and Description='Helps you dare to live your dreams, without fear of failure'.

Hình 2.33: Kết quả thêm một quyển sách mới lần 2

2.6.4.5 Thực hiện Transaction Log Backup Database

```
1 BACKUP LOG [DBMSAssignmentOrderBook]
2 TO DISK = N'/var/opt/mssql/data/DBMSAssignmentOrderBook_Log.trn' WITH NOFORMAT, NOINIT,
3 NAME = N'DBMSAssignmentOrderBook-Log Database Backup', SKIP, NOREWIND, NOUNLOAD, COMPRESSION, STATS = 10
```

Messages

11:19:39 PM Started executing query at Line 1138
37 percent processed.
74 percent processed.
100 percent processed.
Processed 22 pages for database 'DBMSAssignmentOrderBook', file 'DBMSAssignmentOrderBook_Log' on file 1.
BACKUP LOG successfully processed 22 pages in 0.023 seconds (7.302 MB/sec).
Total execution time: 00:00:00.210

Hình 2.34: Kết quả kiểm tra File Transaction Log Backup

2.6.4.6 Restore Database

Bây giờ chúng ta sẽ giả định Database bị sự cố và sẽ tìm cách khôi phục lại từ những File Backup từ trước. Chúng ta sẽ giả định sự cố này bằng cách xoá File Master hiện tại là DBMSAssignmentOrderBook.mdf trong thư mục lưu trữ mssql/data.



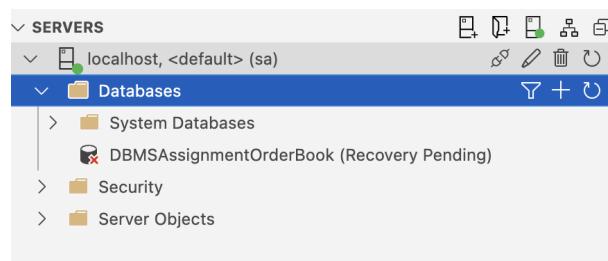
```
$ ls -l
total 154700
-rw-r----- 1 mssql root 593920 Dec 14 16:15 DBMSAssignmentOrderBook.bak
-rw-r----- 1 mssql root 8388608 Dec 14 16:19 DBMSAssignmentOrderBook.ndf
-rw-r----- 1 mssql root 176128 Dec 14 16:17 DBMSAssignmentOrderBook_DIF.DIF
-rw-r----- 1 mssql root 86016 Dec 14 16:19 DBMSAssignmentOrderBook_Log.trn
-rw-r----- 1 mssql root 8388608 Dec 14 16:25 DBMSAssignmentOrderBook_Log.ldf
-rw-r----- 1 mssql root 256 Nov 12 03:14 Entropy.bin
-rw-r----- 1 mssql root 4915200 Dec 14 16:13 master.mdf
-rw-r----- 1 mssql root 2097152 Dec 14 16:22 mastlog.ldf
-rw-r----- 1 mssql root 8388608 Dec 14 16:15 Model.mdf
-rw-r----- 1 mssql root 16056320 Dec 14 16:13 model_msdbdata.mdf
-rw-r----- 1 mssql root 1048576 Dec 14 16:13 model_msdblog.ldf
-rw-r----- 1 mssql root 2097152 Dec 14 16:13 model_replicatedmaster.ldf
-rw-r----- 1 mssql root 4915200 Dec 14 16:13 model_replicatedmaster.mdf
-rw-r----- 1 mssql root 8388608 Dec 14 16:15 modellog.ldf
-rw-r----- 1 mssql root 16056320 Dec 14 16:13 msdbdata.mdf
-rw-r----- 1 mssql root 1310720 Dec 14 16:22 msdblog.ldf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb.mdf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb2.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb3.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb4.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb5.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb6.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb7.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb8.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:21 templog.ldf
```

Hình 2.35: Kết quả kiểm tra File DBMSAssignmentOrderBook.mdf trước khi xoá

```
$ rm DBMSAssignmentOrderBook.mdf
$ ls -l
total 146508
-rw-r----- 1 mssql root 593920 Dec 14 16:15 DBMSAssignmentOrderBook.bak
-rw-r----- 1 mssql root 176128 Dec 14 16:17 DBMSAssignmentOrderBook_DIF.DIF
-rw-r----- 1 mssql root 86016 Dec 14 16:19 DBMSAssignmentOrderBook_Log.trn
-rw-r----- 1 mssql root 8388608 Dec 14 16:31 DBMSAssignmentOrderBook_Log.ldf
-rw-r----- 1 mssql root 256 Nov 12 03:14 Entropy.bin
-rw-r----- 1 mssql root 4915200 Dec 14 16:13 master.mdf
-rw-r----- 1 mssql root 2097152 Dec 14 16:38 mastlog.ldf
-rw-r----- 1 mssql root 8388608 Dec 14 16:15 Model.mdf
-rw-r----- 1 mssql root 16056320 Dec 14 16:13 model_msdbdata.mdf
-rw-r----- 1 mssql root 1048576 Dec 14 16:13 model_msdblog.ldf
-rw-r----- 1 mssql root 2097152 Dec 14 16:13 model_replicatedmaster.ldf
-rw-r----- 1 mssql root 4915200 Dec 14 16:13 model_replicatedmaster.mdf
-rw-r----- 1 mssql root 8388608 Dec 14 16:15 modellog.ldf
-rw-r----- 1 mssql root 16056320 Dec 14 16:13 msdbdata.mdf
-rw-r----- 1 mssql root 1310720 Dec 14 16:22 msdblog.ldf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb.mdf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb2.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb3.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb4.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb5.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb6.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb7.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:13 tempdb8.ndf
-rw-r----- 1 mssql root 8388608 Dec 14 16:31 templog.ldf
```

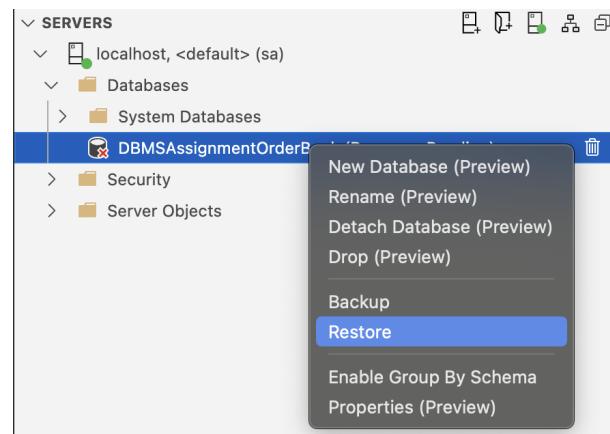
Hình 2.36: Kết quả kiểm tra File DBMSAssignmentOrderBook.mdf sau khi xoá

Khi đó quay lại Database Engine chúng ta sẽ thấy dòng chữ Recover Pending - tức là vì một lý do nào đó làm mất File Master Data khiến nó không có khả năng truy cập được.



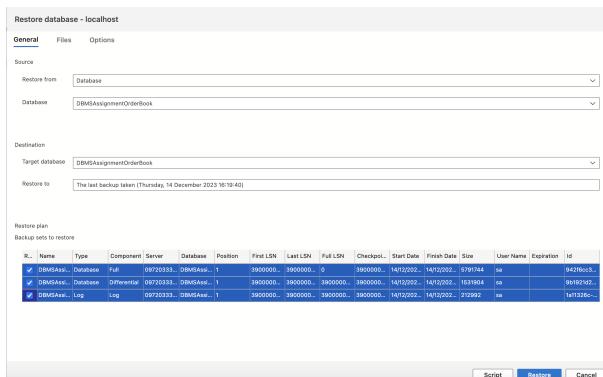
Hình 2.37: Database Engine không thể truy cập được vì sự cố

Bây giờ chúng ta sẽ tiến hành Restore Database từ Gui, nhập chuột phải vào Database bị sự cố và nhấn chọn Restore.



Hình 2.38: Cách vào Restore Database

Một Modal Restore Database hiện ra, ở mục General, phần Source chọn Database là DBMSAssignmentOrderBook. Khi đó ta sẽ chọn tất cả các File Backup chúng ta đã thực hiện ở các bước trên (3 File Backup).



Hình 2.39: Thực hiện chọn các File để Restore Database

Sau đó, chúng ta nhấn nút Restore. Nếu thực hiện thành công thì chúng ta sẽ truy cập được Database .



Hình 2.40: Kết quả Restore thành công

Kiểm tra dữ liệu thì ta vẫn thấy 2 quyển sách được Insert vào trước thời điểm xảy ra sự cố, và các dữ liệu toàn vẹn trước đó.

			Frankenstein	800000	50	400000	2000	Helps you dare to live your dre;
11	11		The Alchemist	800000	50	400000	2000	Helps you dare to live your dre;
12	12		The Alchemist 2	1000000	50	500000	2000	Helps you dare to live your dre;

Hình 2.41: Kiểm tra dữ liệu

CHƯƠNG 3

RAVENDB

3.1 Giới thiệu tổng quan

Thuật ngữ NoSQL được giới thiệu lần đầu vào năm 1998 sử dụng làm tên gọi chung cho các lightweight open source relational database (cơ sở dữ liệu quan hệ nguồn mở nhỏ) nhưng không sử dụng SQL cho truy vấn. Thuật ngữ NoSQL đánh dấu bước phát triển của thế hệ database mới: distributed (phân tán) + non-relational (không ràng buộc). Đây là 2 đặc tính quan trọng nhất.

Sở dĩ người ta phát triển NoSQL suất phát từ yêu cầu cần những database có khả năng lưu trữ dữ liệu với lượng cực lớn, truy vấn dữ liệu với tốc độ cao mà không đòi hỏi quá nhiều về năng lực phần cứng cũng như tài nguyên hệ thống và tăng khả năng chịu lỗi. Đây là những vấn đề mà các relational database không thể giải quyết được. Việc xây dựng cơ sở dữ liệu phi quan hệ có mục đích dành cho các kho dữ liệu phân tán nhằm đáp ứng nhu cầu lưu trữ dữ liệu lớn.

Phân loại NoSQL database. Có bốn loại chung (loại phổ biến nhất) của cơ sở dữ liệu NoSQL gồm Key-value stores, Column-oriented databases, Graph databases, Document Oriented databases. Mỗi loại đều có các thuộc tính và giới hạn riêng. Không có một giải pháp duy nhất nào tốt hơn tất cả các giải pháp khác, tuy nhiên có một số cơ sở dữ liệu tốt hơn để giải quyết các vấn đề cụ thể.

3.1.1 Nguồn gốc và đặc điểm của RavenDB



Hệ quản trị cơ sở dữ liệu RavenDB được tạo bởi Oren Eini, một nhà phát triển đến từ Israel, là một chuyên gia nổi tiếng trong thế giới .Net. Vào tháng 5 năm 2010, Oren phát hành phiên bản 1.0 của RavenDB. Vào tháng 10, quá trình cài đặt đã được bảo mật và sau đó, RavenDB bắt đầu trở nên phổ biến – lên đến thời điểm mà nhà phát triển đạt được 1 triệu lượt tải xuống vào tháng 9 năm 2015. Ngày nay, RavenDB là một cơ sở dữ liệu hoàn thiện và đáng tin cậy, đã được kiểm chứng từ quá trình cài đặt trên các máy nhỏ như Raspberry Pi cho đến các cụm bao gồm hơn một triệu nút.

RavenDB là hệ cơ sở dữ liệu NoSQL hướng tài liệu mã nguồn mở được thiết kế cho .NET và được xây dựng bằng C#. Máy khách .NET được bao gồm để liên lạc với máy chủ nhưng biểu diễn cơ bản là HTTP/JSON - vì vậy, bất kỳ ứng dụng khách nào có thể giao tiếp với máy chủ qua HTTP/JSON đều sẽ hoạt động.

Những lợi ích của RavenDB bên cạnh lợi ích của 1 NoSQL database:

- **Hiệu suất cực cao**

RavenDB được tối ưu hóa cao. Ngay cả trên những chiếc máy như Raspberry Pi 400, bạn cũng sẽ có thể phục vụ hơn 2.000 yêu cầu đọc đồng thời mỗi giây. Phần cứng thương mại sẽ đưa bạn tới 150.000 lần ghi/giây và 1.000.000 lần đọc/giây và tất cả điều đó đều có độ trễ thấp. Hơn nữa, các truy vấn của bạn sẽ liên tục hoạt động trên các chỉ mục được tính toán trước để bạn sẽ nhận được kết quả nhanh chóng.

- **Đầy đủ về giao tác**

Ngay từ đầu, RavenDB đã cung cấp các đảm bảo ACID giao dịch đầy đủ. Các giao dịch đa tài liệu và đa bộ sưu tập cũng được hỗ trợ, cùng với giao dịch trên toàn cụm. Chúng ta sẽ đề cập đến ACID trong các chương sau, nhưng bây giờ, giả sử – ACID là mức tối thiểu mà bất kỳ cơ sở dữ liệu nào cũng phải đảm bảo. Nó sẽ đảm bảo rằng dữ liệu của bạn được không bị mất và cơ sở dữ liệu của bạn duy trì tính nhất quán bất chấp mọi thách thức.

- **Tự động điều chỉnh**

RavenDB là một cơ sở dữ liệu phát triển, biết cách tự chăm sóc bản thân. Nếu bạn cố gắng thực hiện một truy vấn mà không có chỉ mục hỗ trợ nó, RavenDB sẽ tạo một chỉ mục cho bạn. Nếu một nút trong cụm của bạn bị chậm hơn vì lý do nào đó, lưu lượng truy cập sẽ được chuyển hướng đến nút nhanh nhất một cách linh hoạt. Cụm của bạn đang tự giám sát liên tục và liên tục cách thức. Nó sẽ theo dõi các thông số quan trọng như mức sử dụng CPU hoặc mức sử dụng bộ nhớ và hành động theo họ. Nhìn chung, RavenDB đang quan sát môi trường của nó và phản ứng một cách thông minh.

- **An toàn theo mặc định**

An toàn theo mặc định có nhiều khía cạnh và RavenDB tự hào về công nghệ tiên tiến nhất giải pháp kỹ thuật và một tập hợp các giá trị mặc định sẽ cho phép bạn thiết lập và vận hành một cách dễ dàng vài phút thôi. Nó không chỉ là những việc bạn phải làm mà còn là danh sách những việc bạn không làm buộc phải làm để tạo ra các ứng dụng an toàn. Đây là tập hợp con của các tính năng này:

- Mã hóa
- Xác thực
- Giới hạn số lần gọi cơ sở dữ liệu mỗi phiên

- **Tính sẵn có cao**

RavenDB vốn là một cơ sở dữ liệu phân tán. Ngay cả khi bạn chỉ chạy một nút, nó sẽ được coi là một cụm của một nút. Các cụm thường bao gồm một số nút, hầu hết thường là ba. Thiết lập nhiều nút như vậy sẽ cung cấp một số bản sao chính xác của cơ sở dữ liệu và chỉ cần một nút hoạt động thì dữ liệu của bạn sẽ có sẵn.

- **Cấu trúc liên kết**

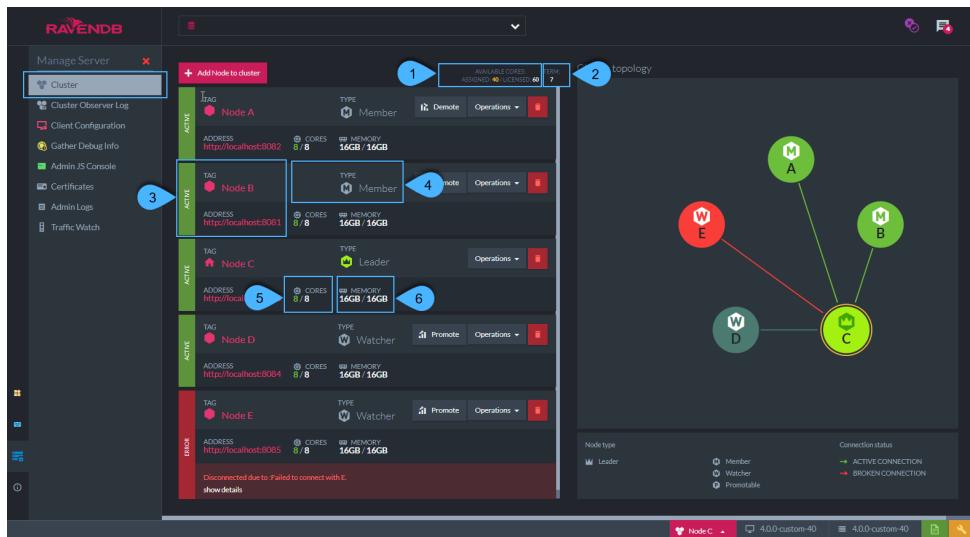
Các cụm RavenDB có thể mở rộng quy mô từ một nút đến vài triệu nút. Thiết lập của bạn có thể bao gồm lưu trữ đám mây, máy cục bộ và tất cả các kiểu sắp xếp không đồng nhất. Cuối cùng, các cấu trúc liên kết hình ngôi sao khác nhau được hỗ trợ với vị trí trung tâm (minh họa trong hình 3.1) thực hiện sao chép hoàn chỉnh hoặc được lọc hai chiều với hàng triệu vị trí biên.

3.1.2 Tổ chức dữ liệu

3.1.2.1 Cluster và Node

Một nhóm máy chủ RavenDB có thể có hoặc không có trên cùng một thiết bị. Nhóm máy chủ này cho phép các operations trên toàn Cluster (cụm) thực thi trên mỗi Node, sử dụng Raft để điều phối việc thực thi. Nếu có Node dẫn đầu, Cluster đảm bảo rằng ít nhất $(n/2) + 1$ Node sẽ có các operations thực hiện trên chúng.

Một Node RavenDB là thành viên của Cluster. Với tư cách là thành viên của cụm, nó thực thi các operations trên toàn cụm, được điều khiển thông qua Raft Commands. Một Cluster bao gồm một hoặc nhiều phiên bản máy chủ RavenDB được gọi là các Node.



Hình 3.1: Cluster và Node trong RavenDB

3.1.2.2 Document Store - Document Session

Giao tiếp giữa máy khách và máy chủ trong RavenDB được thực hiện qua HTTP. Đối tượng DocumentStore chịu trách nhiệm xử lý toàn bộ quá trình liên lạc cũng như cung cấp các dịch vụ bổ sung như bộ nhớ đệm. Về cơ bản, đó là đường dẫn giữa ứng dụng của bạn và máy chủ cơ sở dữ liệu.

Khởi tạo DocumentStore trên máy khách. DocumentStore được định cấu hình để giao tiếp với máy chủ RavenDB cục bộ mà bạn đã chạy dưới dạng bảng điều khiển và sử dụng cơ sở dữ liệu DBMS_RavenDB đã tạo trước đó:

```

1  var docStore = new DocumentStore
2    {
3      Url = "http://localhost:8080",
4      DefaultDatabase = "DBMS_RavenDB",
5    };
6  docStore.Initialize();

```

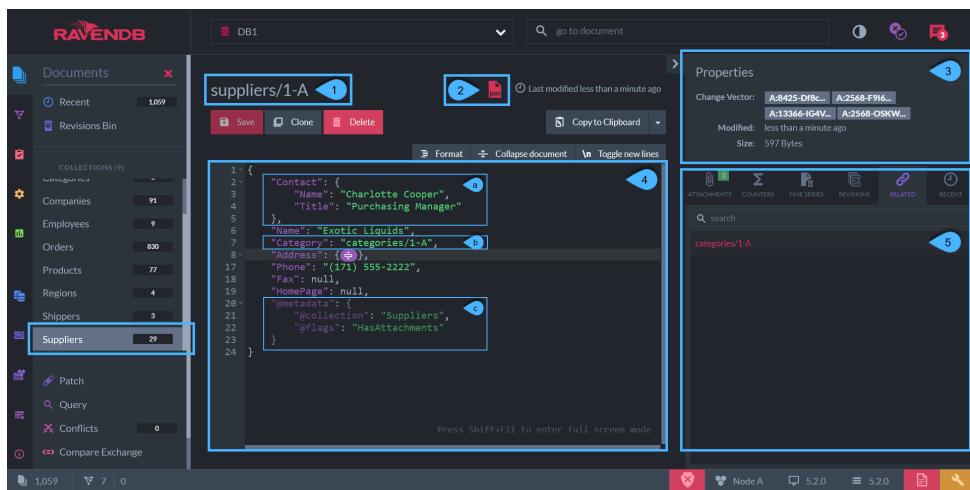
Sau khi khởi tạo đối tượng DocumentStore, kết nối giữa ứng dụng của bạn và máy chủ RavenDB sẽ được thiết lập. Ta có thể sử dụng nó để tạo các đối tượng DocumentSession, gọi đơn giản là phiên. Phiên là công cụ chính của bạn cho hầu hết các hoạt động ở phía Client. Từ đó ta có thể giao tiếp với máy chủ và thực hiện các thao tác đọc hoặc ghi đổi với nó. Để tạo một phiên, chỉ cần gọi DocumentStore.OpenSession().

3.1.2.3 Document

RavenDB đang lưu trữ tài liệu ở định dạng JSON và chúng gần như không giới hạn kích thước – chúng có thể tăng lên tới 2Gb. Tuy nhiên, bạn nên giới hạn kích thước tài liệu của mình ở mức vừa phải, không lớn hơn vài megabyte. JSON lớn hơn thế sẽ khó tải và lưu lại vào cơ sở dữ liệu và thường là tín hiệu mạnh mẽ cho thấy mô hình của bạn chưa tối ưu.

Bằng cách sử dụng .NET Client API, bất kỳ lớp nào trong ứng dụng của bạn đều có thể được tuân tự hóa thành JSON và được lưu trữ dưới dạng tài liệu RavenDB.

Mọi loại dữ liệu đều được hỗ trợ, từ các loại gốc, chẳng hạn như chuỗi và loại số, cho đến ngày tháng và đối tượng của các lớp khác. Biểu đồ đối tượng hoàn chỉnh được lưu vào RavenDB dưới dạng một tài liệu JSON. Mỗi tài liệu có một mã định danh duy nhất, do đó nó có thể được xác định duy nhất và được lấy từ cửa hàng.



Hình 3.2: Document và Collection trong RavenDB

3.1.2.4 Collection

Bởi vì các tài liệu trong RavenDB không được nhóm theo bất kỳ cách nào—tất cả chúng đều được lưu trữ trong cùng một không gian lưu trữ và không có gì thực sự phân biệt chúng—nên cần có cách nào đó để nhóm các tài liệu tương tự lại với nhau và nói về các nhóm trong số chúng.

Để đạt được mục đích đó, RavenDB cho phép đóng dấu một tài liệu bằng một giá trị chuỗi sẽ là bằng chứng về loại của nó. RavenDB thường muốn giá trị đó mang tính mô tả nhiều nhất có thể, chẳng hạn như “Người dùng” và “Sản phẩm”.

Khi sử dụng .NET Client API, việc này được thực hiện tự động cho bạn. Khi lưu một đối tượng vào RavenDB, dấu loại sẽ được lưu cùng với tài liệu sẽ được suy ra từ tên lớp đối tượng. Quy ước là sử dụng dạng số nhiều của tên lớp, do đó, đối tượng Người dùng sẽ được tuân tự hóa thành tài liệu JSON và sau đó được đóng dấu bằng loại thực thể “người dùng” và đối tượng Sản phẩm sẽ có tem loại “sản phẩm” trên đó.

Điều này giúp RavenDB quyết định xem collection nào có trong cơ sở dữ liệu—bằng cách tra cứu tất cả các tem loại duy nhất trong cơ sở dữ liệu. Một collection trong RavenDB chỉ là một cách để nói về tất cả các tài liệu có chung tem loại, giống như việc bạn đề cập đến các sản phẩm gỗ trông giống như những chiếc ghế là “những chiếc ghế”. Ví dụ: bộ sưu tập Sản phẩm về cơ bản là một yêu cầu từ cơ sở dữ liệu để hiển thị tất cả các tài liệu có tem loại “sản phẩm”. Số lượng collection trong phiên bản RavenDB bằng số lượng giá trị duy nhất của các loại tem đó.

Điều quan trọng cần nhớ là collection trong RavenDB chỉ là một cách hợp lý để nói về tất cả các tài liệu có cùng loại. Đó là một khái niệm hoàn toàn ảo và khác hoàn toàn với bảng SQL.

3.1.2.5 Mã định danh duy nhất

Về cơ bản, RavenDB chỉ là một kho lưu trữ khóa/giá trị để xử lý các tài liệu. Một tài liệu được lưu trữ với một khóa duy nhất để nhận dạng nó, khóa này cũng được sử dụng để lấy nó ra khỏi bộ lưu trữ sau này. Trong RavenDB, các khóa đó, còn được gọi là ID tài liệu, là các chuỗi đơn giản. Bất kỳ chuỗi nào cũng có thể được sử dụng làm khóa tài liệu và chuỗi đó là duy nhất trên tất cả các tài liệu trong hệ thống, bất kể chúng thuộc bộ sưu tập nào.

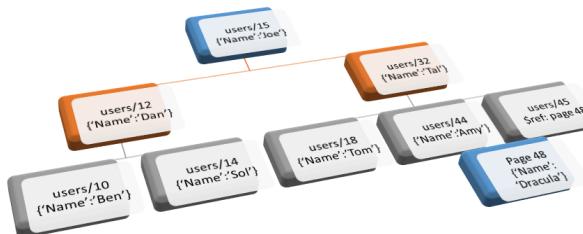
RavenDB giúp bạn không phải chịu trách nhiệm theo dõi các ID duy nhất trong cùng bộ sưu tập. Bằng cách sử dụng .NET Client API, bất cứ khi nào bạn yêu cầu lưu trữ một đối tượng mới, RavenDB sẽ gán cho nó một ID duy nhất bằng cách tạo tiền tố và sẽ thương lượng một ID số nguyên duy nhất chưa được sử dụng trong bộ sưu tập.

3.1.3 Lưu trữ dữ liệu

RavenDB sử dụng công cụ lưu trữ được quản lý nội bộ có tên Voron để duy trì dữ liệu (tài liệu, chỉ mục và cấu hình). Đó là một công cụ lưu trữ hiệu suất cao được thiết kế và tối ưu hóa theo nhu cầu của RavenDB. Nó sử dụng các cấu trúc bên dưới sau để cho phép tổ chức dữ liệu trên bộ lưu trữ liên tục một cách hiệu quả:

- B+Tree - Khóa và giá trị có kích thước thay đổi.
- Fixed Sized B+Tree - khóa Int64 và giá trị kích thước cố định (được xác định tại thời điểm tạo). Nó cho phép bạn tận dụng nhiều tính năng tối ưu hóa khác nhau.
- Raw data section (Phần dữ liệu thô) – Cho phép bạn lưu trữ dữ liệu thô (ví dụ: nội dung tài liệu) và cung cấp mã định danh cho phép truy cập dữ liệu trong thời gian O(1). Thay vì lưu trữ dữ liệu trực tiếp trên cây, chúng ta sẽ chuyển dữ liệu vào bộ lưu trữ riêng. Phần dữ liệu thô có thể nhỏ (đối với các giá trị có kích thước nhỏ hơn 2KB) hoặc lớn. Các giá trị nhỏ được nhóm lại với nhau thành các phần có độ dài từ 2 – 8 MB. Mặc dù mỗi giá trị có kích thước lớn hơn sẽ được đặt độc lập, nhưng được làm tròn đến kích thước trang tiếp theo (4KB, theo mặc định).

Các giá trị nhỏ có thể được lưu trữ trực tiếp bên trong B+Tree, trong khi các giá trị lớn hơn yêu cầu lưu trữ riêng và cây tài liệu chỉ lưu trữ tham chiếu đến dữ liệu (raw data section id) trên đĩa (thực tế là vị trí tệp cho dữ liệu).



Hình 3.3: Minh họa cây tài liệu - Các giá trị nhỏ

- Table (Bảng) - Sự kết hợp của các Phần dữ liệu thô với bất kỳ số lượng chỉ mục nào là Cây B+Tree có kích thước thay đổi hoặc cố định.

Phần dữ liệu thô cho phép ta chỉ cần yêu cầu nó lưu trữ một loạt dữ liệu và có được id (int64) tương ứng với chi phí O(1) để truy cập các giá trị đó bằng id. Sau đó, sử dụng id này làm giá trị trong B+Tree, có nghĩa là cấu trúc tổ chức dữ liệu trông như thế này:

- Raw data sections – [document json, document json, etc]
- Documents tree (B+Tree) - (key: document id, value: raw data section id)
- Etags tree (B+Tree) - (key: etag, value: raw data section id)

3.2 Indexing

Trong RavenDB, máy chủ đang sử dụng Lucene để thực hiện lập chỉ mục và Ngôn ngữ truy vấn Raven (RQL) để truy vấn.

Các chỉ mục được chia thành nhiều loại:

- Auto Indexes -vs- Static Indexes
- Map Indexes -vs- Map-Reduce Indexes
- Single-Collection Indexes -vs- Multi-Collection Indexes

Ngoài ra, còn có một số chỉ mục khác như Fanout Index, Multi-Map Index...

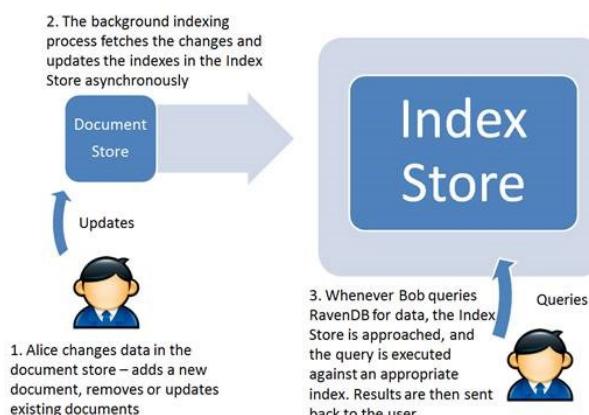
Indexed Data

- Index Field: trường được lập chỉ mục
- Index Entries: tất cả các trường và giá trị đã được lập chỉ mục từ một tài liệu cụ thể. Thông thường, nó sẽ là một tập hợp con của các trường trong tài liệu đang được lập chỉ mục, nhưng nó cũng có thể là một số trường được tính toán.
- Term:
 - Các giá trị Index Entries được chia thành các Term theo Analyzer được chỉ định được sử dụng trong định nghĩa chỉ mục.
 - Là giá trị được lập chỉ mục thực tế được lưu trữ trong chỉ mục. Giá trị này thường giống với giá trị của trường được lập chỉ mục nhưng có thể khác nếu bạn đang áp dụng full-text-search.

Cách hoạt động của chỉ mục là lặp lại các tài liệu và xây dựng hàm Map, ánh xạ giữa các Term được lập chỉ mục và các tài liệu thực tế chứa chúng. Sau lần lập chỉ mục đầu tiên, chỉ mục sẽ giữ cho hàm Map luôn cập nhật khi các thao tác cập nhật hay thêm mới diễn ra trong cơ sở dữ liệu.

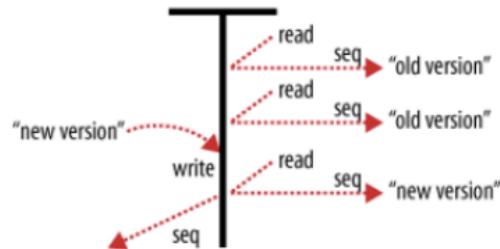
3.2.1 Quá trình lập chỉ mục

RavenDB có một quy trình nền sẽ chuyển các tài liệu mới và cập nhật tài liệu khi chúng được đưa vào, ngay sau khi chúng được lưu trữ trong Kho Tài liệu và chuyển chúng theo đợt qua tất cả các chỉ mục trong hệ thống. Đối với các thao tác ghi, người dùng sẽ nhận được xác nhận ngay lập tức về giao dịch, ngay cả trước khi quá trình lập chỉ mục bắt đầu xử lý các cập nhật này mà không cần chờ lập chỉ mục, nhưng chắc chắn 100% các thay đổi đã được ghi lại trong cơ sở dữ liệu. Các truy vấn cũng không chờ lập chỉ mục; họ chỉ sử dụng các chỉ mục tồn tại tại thời điểm truy vấn được đưa ra. Điều này được thể hiện trong hình sau đây.



Hình 3.4: Quá trình lập chỉ mục

Vì việc lập chỉ mục được thực hiện ở chế độ nền nên khi có đủ dữ liệu, quá trình đó có thể mất một lúc để hoàn thành. Điều này có nghĩa là có thể mất một lúc để tài liệu mới xuất hiện trong kết quả truy vấn. Mặc dù RavenDB được tối ưu hóa cao độ để giảm thiểu những trường hợp như vậy nhưng điều đó vẫn có thể xảy ra và khi điều này xảy ra, kết quả chỉ mục đã cũ.



Hình 3.5: Chỉ mục cũ

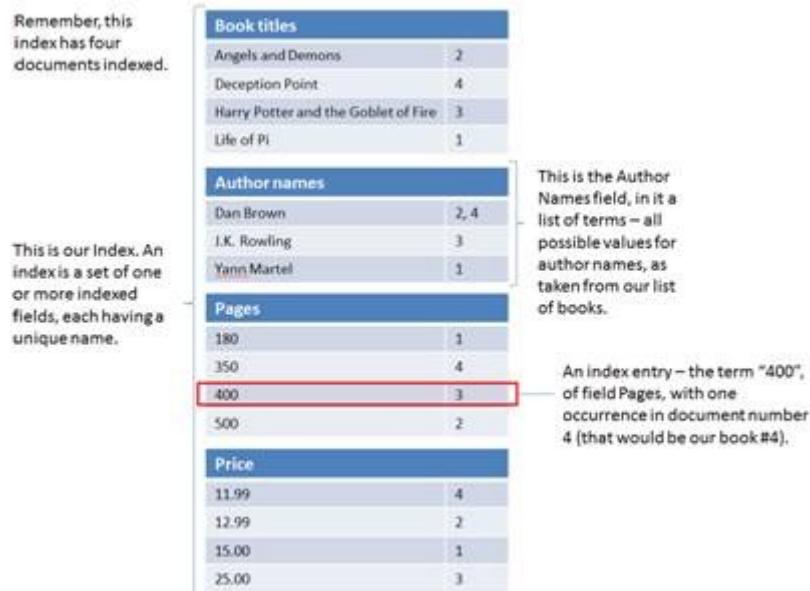
RavenDB sử dụng Lucene.NET làm cơ chế lập chỉ mục. Lucene.NET là công cụ tìm kiếm nguồn mở phổ biến Lucene. Được viết ban đầu bằng Java và được phát hành lần đầu tiên vào năm 2000, Lucene là thư viện công cụ tìm kiếm nguồn mở hàng đầu. Nó đang được các trang web lớn như Twitter, LinkedIn, v.v. sử dụng để làm cho nội dung của họ có thể tìm kiếm được và liên tục được cải tiến để làm cho nội dung của họ nhanh hơn và tốt hơn.

3.2.2 Lucene Indexes

Bởi vì các chỉ mục RavenDB trên thực tế là các chỉ mục Lucene, nên trước khi tìm hiểu sâu hơn về các chỉ mục RavenDB, ta cần biết một số khái niệm về Lucene.

Trong Lucene, thực thể cơ sở đang được lập chỉ mục được gọi là document và mọi tìm kiếm đều mang lại danh sách các tài liệu phù hợp. Trong ví dụ này bạn đang tìm kiếm sách nên mỗi cuốn sách sẽ là một tài liệu Lucene. Giống như sách có tựa đề, tác giả, tên, số trang, v.v., đôi khi bạn cần có khả năng tìm kiếm các tài liệu có nhiều hơn một phần dữ liệu được lấy từ mỗi tài liệu. Để đạt được mục đích này, mọi tài liệu trong Lucene đều có khái niệm về các trường, đây chỉ là sự phân tách logic giữa các phần khác nhau của tài liệu mà bạn đã lập chỉ mục. Mỗi trường trong tài liệu Lucene có thể chứa các phần thông tin khác nhau về tài liệu mà sau này bạn có thể sử dụng để tìm kiếm.

Để áp dụng những khái niệm đó vào ví dụ về sách của chúng ta, trong Lucene mỗi cuốn sách sẽ là một tài liệu và mỗi tài liệu sách như vậy sẽ có các trường Tiêu đề, Tên tác giả, Giá và Số trang. Lucene sẽ tạo một chỉ mục với một số danh sách giá trị, mỗi danh sách cho mỗi trường, giống như trong hình dưới đây. Để hoàn thiện bức tranh, mỗi giá trị được đưa vào chỉ mục ở một trong các danh sách đó (ví dụ: Dan Brown từ hai cuốn sách trong trường Tên tác giả) được gọi là một Term.



Book titles	
Angels and Demons	2
Deception Point	4
Harry Potter and the Goblet of Fire	3
Life of Pi	1
Author names	
Dan Brown	2, 4
J.K. Rowling	3
Yann Martel	1
Pages	
180	1
350	4
400	3
500	2
Price	
11.99	4
12.99	2
15.00	1
25.00	3

Hình 3.6: Minh họa Lucene Indexes

Các tìm kiếm được thực hiện với các Term và tên trường để tìm tài liệu phù hợp, trong đó tài liệu được coi là khớp nếu nó có các thuật ngữ được chỉ định trong các trường được tìm kiếm, giống như truy vấn giả sau: tất cả các tài liệu Sách có trường Tác giả có giá trị Dan Brown. Lucene cho phép truy vấn với nhiều mệnh đề trên cùng một trường hoặc thậm chí trên các trường khác nhau, vì vậy các truy vấn như “tất cả sách của tác giả Dan Brown hoặc JK Rowling và có giá thấp hơn 50 đô la” đều được hỗ trợ đầy đủ.

Một chỉ mục trong RavenDB chỉ là một chỉ mục Lucene tiêu chuẩn. Mọi tài liệu RavenDB từ Cửa hàng Tài liệu đều có thể được lập chỉ mục bằng cách tạo tài liệu Lucene từ đó. Khi đó, một trường trong tài liệu Lucene đó sẽ là một phần có thể tìm kiếm được của tài liệu RavenDB mà bạn đang lập chỉ mục—ví dụ: tiêu đề của một bài đăng blog, nội dung thực tế và ngày đăng, mỗi trường sẽ là một trường.

Quá trình lập chỉ mục trong đó các tài liệu Lucene được tạo từ các tài liệu được lưu trữ trong RavenDB—from JSON có cấu trúc thô đến cấu trúc phẳng của các tài liệu và trường—sử dụng hai loại hàm được gọi là Map và Reduce, được khai báo cho mọi chỉ mục (trong đó hàm Reduce là tùy chọn).

3.2.3 Map Indexes

Đây là phần cơ bản nhất của chỉ mục. Tất cả các chỉ mục đều có hàm Map và đây là thao tác đầu tiên chạy trên mọi tài liệu được chuyển tới chỉ mục.

Map Index chứa một (hoặc nhiều) hàm ánh xạ cho biết trường nào trong tài liệu sẽ được lập chỉ mục. Chúng cho biết tài liệu nào có thể được tìm kiếm theo trường nào. Các hàm ánh xạ này là các hàm dựa trên LINQ hoặc các hàm JavaScript (khi sử dụng chỉ mục JavaScript) và có thể coi là cốt lõi của chỉ mục.

Sử dụng ví dụ về hiệu sách, việc ánh xạ một tài liệu tới một mục nhập chỉ mục sẽ trông giống như hình sau. Kết quả chỉ là một chỉ mục Lucene với bốn trường, giống hệt như trường. Sau khi bạn trích xuất các giá trị từ tài liệu JSON được lưu trữ trong RavenDB và đặt các mục nhập chỉ mục cho nó vào các trường tương ứng, nó sẽ trở thành thứ bạn có thể truy vấn.



```
{  
    "document_id": 1,  
    "title": "Life of Pi",  
    "author": "Yann Martel",  
    "pages": 180,  
    "price": 15.00  
}
```

Book titles	
...	1
Life of Pi	1
...	1
Author names	
...	1
Yann Martel	1
...	1
Pages	
...	1
180	1
...	1
Price	
...	1
15.00	1
...	1

Sử dụng RQL để thể hiện việc trích xuất hoặc ánh xạ dữ liệu này. Tạo index tên BookShortDetail cho collection Books:

```
1   from doc in docs.Books  
2   /* docs represents the entire collection of documents stored in the database, and docs.Books tells RavenDB  
3   to look at only documents in the Books collection. */  
4   select new { Title = doc.Title, Author = doc.Author, Pages = doc.Pages, Price = doc.Price }  
5   --The Map function outputs an anonymous object with properties as defined within the curly brackets.
```

Quy ước: Cú pháp trong Studio gồm docs.Books khác với khi viết trong ứng dụng, chỉ gồm Books

Điều này yêu cầu RavenDB chỉ xem xét các tài liệu của bộ sưu tập Sách và ánh xạ các thuộc tính Tiêu đề, Tác giả, Trang và Giá từ mọi tài liệu được cung cấp cho chỉ mục với các tên tương ứng đó.

Dầu ra của hàm Bản đồ là một tập hợp các đối tượng ẩn danh có cấu trúc phẳng - trong trường hợp này là một đối tượng có các trường Tiêu đề, Tác giả, Trang và Giá. Mỗi đối tượng đó sẽ được đưa vào chỉ mục, mỗi trường của nó trong một trường chuyên dụng trong Lucene.

Trong ví dụ trên, where có thể được sử dụng trong truy vấn để lọc kết quả. Nếu bạn luôn muốn lọc với cùng điều kiện lọc, bạn có thể sử dụng where trong định nghĩa chỉ mục để thu hẹp các thuật ngữ chỉ mục mà truy vấn phải quét. Điều này có thể tiết kiệm thời gian truy vấn nhưng thu hẹp các thuật ngữ có sẵn để truy vấn.

Ví dụ, truy vấn để lọc kết quả, trích xuất các sách có tác giả là Yann Martel:

```
1   from index 'Books/BookShortDetail' where Author = 'Yann Martel'
```

Map Index tên 'Books/ByAuthor' ánh xạ các thuộc tính Tiêu đề, Tác giả, Trang và Giá từ mọi tài liệu được cung cấp cho chỉ mục, chỉ với điều kiện Tác giả là Yann Martel.



```
1 from doc in docs.Books
2 where doc.Author = 'Yann Martel'
3 select new [ Title = doc.Title, Author = doc.Author, Pages = doc.Pages, Price = doc.Price ]
```

3.2.4 Map Reduce Indexes

Các chỉ mục Map-Reduce cho phép tổng hợp dữ liệu phức tạp, có thể được truy vấn với chi phí rất thấp, bất kể kích thước dữ liệu. Để đẩy nhanh các truy vấn và ngăn chặn tình trạng suy giảm hiệu suất trong quá trình truy vấn, việc tổng hợp được thực hiện trong giai đoạn lập chỉ mục chứ không phải tại thời điểm truy vấn.

Trong SQL Server bạn có thể sử dụng phương thức của Aggregation như COUNT, GROUP BY.... Nhưng một truy vấn như vậy sẽ thực hiện các tính toán khi được thực thi. Tuy nhiên, với RavenDB, Query Optimizer sẽ hỗ trợ tự động tạo các Map Reduce Index trong giai đoạn lập chỉ mục, không phải khi truy vấn. Khi dữ liệu mới vào cơ sở dữ liệu hoặc tài liệu hiện có được sửa đổi, chỉ mục Map-Reduce sẽ tính toán lại dữ liệu tổng hợp để kết quả tổng hợp luôn có sẵn và cập nhật.

Map/Reduce chỉ là một tên phức tạp cho một thao tác hai giai đoạn đơn giản để chọn và nhóm dữ liệu, được xác định bởi một cặp hàm: hàm Map và hàm Giảm. Thao tác này cho phép bạn thực hiện các tính toán trong quá trình lập chỉ mục và sau đó làm cho kết quả có thể truy cập được để tìm kiếm.

Phần đầu tiên của bất kỳ hoạt động Map Reduce nào là giai đoạn Map. Đó chính xác là quy trình tương tự như phần trước. Dữ liệu đang được trích xuất từ tài liệu bằng hàm Map, nhưng lần này, thay vì lưu trữ nó trong chỉ mục như bạn làm trong chỉ mục Map đơn giản, bạn làm cho nó trải qua một giai đoạn bổ sung, được gọi là giai đoạn Reduce.

Giai đoạn Reduce là nơi diễn ra tính toán nói trên. Nó bao gồm việc nhóm dữ liệu được thu thập bởi giai đoạn Map thành các nhóm dựa trên một số khóa, thường là thứ gì đó liên quan đến dữ liệu từ chính tài liệu, được gọi là khóa rút gọn. Sau đó, mỗi nhóm chứa các tài liệu khớp với khóa rút gọn và bạn có thể thực hiện thao tác tổng hợp trên các nhóm đó. Kết quả là giá trị kết quả cho mỗi nhóm cùng với giá trị của khóa rút gọn.

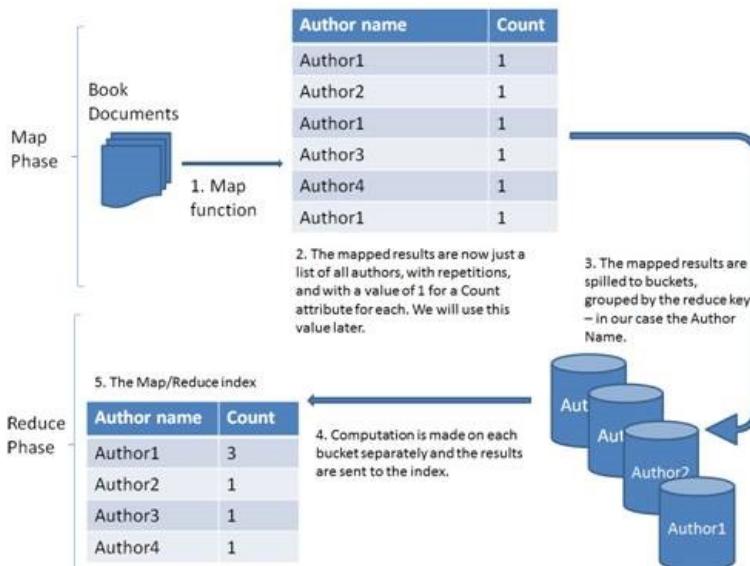
Bằng cách ghi kết quả của giai đoạn Reduce vào chỉ mục, bạn có thể truy vấn các kết quả được tính toán trước của phép tính đó thay vì tính toán nó trong khi truy vấn.

Ví dụ, truy vấn “tất cả các tác giả đã xuất bản năm cuốn sách trở lên”. Bởi vì bạn không có số lượng sách do mỗi tác giả viết trong tài liệu sách của bạn hoặc trong bất kỳ tài liệu nào khác về vấn đề đó, nên bạn cần có chỉ mục Map/Reduce để trả lời truy vấn này. Để làm điều đó, chỉ mục cần chứa tất cả tên tác giả cùng với số lượng sách họ đã viết, do đó, truy vấn về số lượng sách sẽ cung cấp cho bạn tên tác giả.

Bước đầu tiên là thu thập dữ liệu bạn có. Bạn có thể sử dụng hàm Map để duyệt qua tất cả tài liệu Sách mà bạn có và liệt kê tất cả tên tác giả mà bạn có sách. Vì mỗi tác giả được lựa chọn đại diện cho một cuốn sách họ đã viết nên bạn sẽ gán cho bản ghi đó số lượng dữ liệu bổ sung là 1:

```
1 from doc in docs.Books
2 select new [ AuthorName = doc.Author, Count = 1 ]
```

Kết quả của hàm Map sẽ không được ghi vào chỉ mục mà sẽ được lưu trữ dưới dạng kết quả được ánh xạ trong bộ lưu trữ dành riêng cho chỉ mục này. Đầu ra của hàm Map trong chỉ mục Map/Reduce là một tài liệu nhỏ hơn, chứa một tập hợp con dữ liệu từ tài liệu gốc, có thể có một số bổ sung—như thuộc tính Count trong trường hợp này. Dữ liệu trong tập hợp con này sẽ có thể tìm kiếm được hoặc sẽ được yêu cầu để tính toán ở bước tiếp theo (hoặc cả hai).



Hình 3.7: Quy trình trong Map Reduce Index

Bước tiếp theo là giảm tất cả các bản ghi trong bộ lưu trữ kết quả được ánh xạ cho chỉ mục này dựa trên khóa rút gọn. Khóa rút gọn được chọn theo phần dữ liệu bạn muốn nhóm mọi thứ theo đó. Trong trường hợp này, bạn muốn đếm số lần xuất hiện của mỗi tác giả, do đó bạn nhóm theo tên tác giả.

Ở giai đoạn này, bạn có các nhóm chứa số lượng tác giả duy nhất. Mỗi nhóm được đánh dấu bằng một giá trị duy nhất cho khóa rút gọn về cơ bản là tên tác giả và chứa tất cả các kết quả được ánh xạ khớp với giá trị đó cho khóa rút gọn. Bây giờ bạn có thể áp dụng tính toán trên từng nhóm riêng biệt. Tính toán của bạn sẽ là đếm số lượng đối tượng mà nó có, được thực hiện bằng cách tổng hợp thuộc tính Count của tất cả các kết quả được ánh xạ trong mỗi nhóm. Sau đó, bạn sẽ lấy giá trị cho từng nhóm cùng với kết quả tính toán và đó sẽ là kết quả cuối cùng của chỉ mục Map/Reduce của bạn.

Hàm Reduce:

```

1 from result in results
2 group result by result.AuthorName into g
3 select new { AuthorName = g.Key, Count = g.Sum(x => x.Count) }

```

3.2.5 Auto Indexes

Để giảm bớt quá trình phát triển, RavenDB sử dụng các chỉ mục tự động, giúp bạn giảm bớt gánh nặng khai báo trước các chỉ mục theo cách thủ công. Quản lý tự động các chỉ mục là một phần trong tính năng tự động điều chỉnh của RavenDB, cho phép bạn—nhà phát triển—tập trung vào những gì quan trọng nhất và để lại các vấn đề kỹ thuật cho cơ sở dữ liệu.

Khi truy vấn RavenDB để tìm dữ liệu, bạn đang truy vấn một chỉ mục cụ thể trong RavenDB. Tên của chỉ mục cần truy vấn có thể được chỉ định trong chính truy vấn đó, cho RavenDB biết rằng truy vấn này cần được thực thi dựa trên chỉ mục mang tên này.

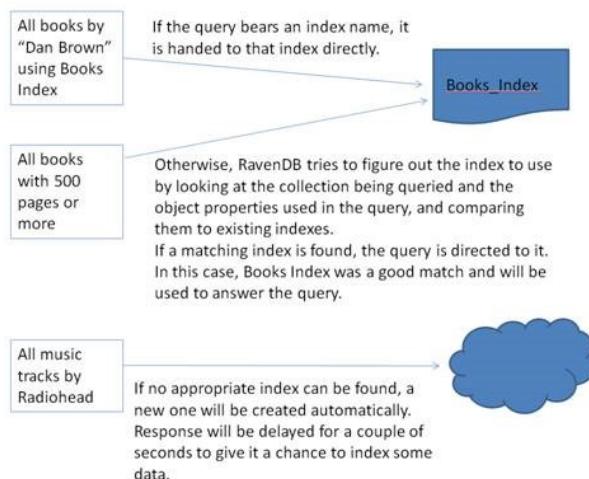
Tuy nhiên, việc chỉ định tên chỉ mục trong truy vấn không phải lúc nào cũng bắt buộc. Đối với hầu hết các truy vấn phổ biến, RavenDB đủ thông minh để tự mình tìm ra chỉ mục nào có thể được sử dụng để trả lời chúng. Nếu một truy vấn đã chỉ định rõ ràng tên chỉ mục, RavenDB sẽ sử dụng chỉ mục cụ thể đó để trả lời truy vấn; nếu không, RavenDB sẽ cố gắng tự động chọn một chỉ mục dựa trên nội dung đang được truy vấn.

Việc tạo chỉ mục tự động sẽ chỉ xảy ra khi một truy vấn không có hướng dẫn sử dụng một chỉ mục cụ thể được thực thi và chỉ khi RavenDB không thể tìm thấy chỉ mục hiện có để đáp ứng truy vấn. Trong trường hợp này, **Máy chủ Query Optimizer** sẽ tạo một chỉ mục thích hợp hoặc mở rộng một chỉ mục

hiện có để nó cũng có thể trả lời loại truy vấn này. Các chỉ mục như vậy được gọi là chỉ mục tự động và được RavenDB quản lý hoàn toàn.

Các chỉ mục Tự động được RavenDB xác định bằng tiền tố đặt tên, Auto/ sau đó là tên bộ sưu tập và thuật ngữ lọc được yêu cầu trong truy vấn.

Ví dụ cho quá trình lựa chọn chỉ mục tự động và tạo chỉ mục tự động:



Giả sử, có một chỉ mục trên bộ sưu tập Sách là Books_Index. Một truy vấn yêu cầu "all books by Dan Brown using Books_Index" sẽ được định tuyến trực tiếp đến Books_Index vì bạn đã chỉ định rõ ràng tên chỉ mục. Sau đó, truy vấn sẽ được chỉ mục đó xử lý và kết quả sẽ được gửi ngay lập tức khi chúng quay trở lại từ chỉ mục.

Xem xét truy vấn "all books with 500 pages or more". Hãy lưu ý cách nó không chỉ định chỉ mục nào sẽ sử dụng. Vì tất cả tài liệu trong bộ sưu tập Sách đều được lập chỉ mục và bạn cũng đảm nhiệm việc lập chỉ mục dữ liệu số trang chứa trong đó vào một trường trong Books_Index, nên bạn biết rằng chỉ mục đó có thể trả lời truy vấn này cho bạn. Đây chính xác là những gì RavenDB sẽ làm: bằng cách xem xét các chỉ mục bạn có trong cơ sở dữ liệu của mình, nó sẽ phát hiện ra rằng Books_Index đã lập chỉ mục tất cả các tài liệu Sách và số trang của chúng cũng được lập chỉ mục.

Truy vấn "all music tracks by Radiohead", vì không có chỉ mục phù hợp được tìm thấy nên Query Optimizer sẽ tạo 1 chỉ mục cho truy vấn đó 1 cách tự động.

Mình họa cho trường hợp hệ thống tạo chỉ mục tự động. Thực hiện truy vấn tất cả nhân viên có địa chỉ thành phố ở London:

```
1 from Employees where Address.City = 'London'
```

Giả sử, không có chỉ mục phù hợp được tìm thấy, RavenDB tạo và chỉ định chỉ mục Auto/Employees/ByAddress.City cho truy vấn, sử dụng trình phân tích "Exact". Thời gian có được kết quả là 62ms, gồm thời gian tìm chỉ mục phù hợp và tạo mới.

The screenshot shows the RavenDB Studio interface. At the top, there is a 'Query' tab with syntax highlighting for a C#-like query: 'From Employees where Address.City = 'London''. Below it is a 'Results' tab displaying a table of employee records from an index named 'Index 'Auto/Employees/ByAddress.City''. The table has columns: Preview, Id, LastName, FirstName, Title, Address, HiredAt, and Birthday. The results show four employees: Dodsworth, Buchanan, Suyama, and King.

Hình 3.8: Kết quả truy vấn trong Studio

Khi chạy truy vấn 1 lần nữa, thời gian thực thi bằng 0ms vì lúc này, chỉ mục được phát hiện ngay lập tức và đưa kết quả cho người dùng.

This screenshot shows the 'Results' tab again, but this time the results are displayed much faster. A message at the top says 'Index 'Auto/Employees/ByAddress.City' was used to get the results in 0 ms'. The table structure and data are identical to the previous screenshot.

Hình 3.9: Kết quả truy vấn lần thứ 2

3.2.6 Static Indexes

Tạo chỉ mục theo cách thủ công có nghĩa là viết các hàm Map và Reduce bằng tay rồi chuyển định nghĩa chỉ mục đến máy chủ để đăng ký. Khi chỉ mục đã được đăng ký trên máy chủ, RavenDB sẽ bắt đầu lập chỉ mục toàn bộ tập dữ liệu trong đó.

Các chỉ mục tĩnh bắt đầu bằng tên được đặt cho chúng khi chỉ mục được xác định theo cách thủ công.

Tùy chọn trường chỉ mục: RavenDB cho phép xác định các tùy chọn khác nhau trên các trường riêng lẻ được lập chỉ mục trong quá trình lập chỉ mục.

Các tùy chọn trên trường cơ bản:

This screenshot shows the 'SpatialData' index configuration screen. It includes sections for 'Maps' and 'Reduce' logic, field definitions, and spatial settings. A yellow callout box highlights the 'CreateSpatialField' method in the Map logic, with the text 'Use 'CreateSpatialField' to create 'Coordinates', the Spatial Index-Field'. Another yellow callout points to the 'Coordinates' field definition in the Fields section. The spatial settings at the bottom include 'Type: Geography', 'Radius Units: Kilometers', and coordinate ranges for Min X, Min Y, Max X, and Max Y.

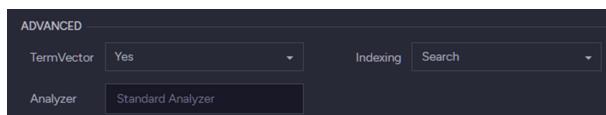
Hình 3.10: Các tùy chọn cơ bản hiển thị trong Studio

- Store, chịu trách nhiệm về cách dữ liệu được tải trả lại kết quả của truy vấn. Khi RavenDB thực thi một truy vấn bằng chỉ mục, theo mặc định, chỉ mục đó chỉ trả về danh sách ID tài liệu khớp với truy vấn. RavenDB sau đó sử dụng các ID đó để tải tài liệu trước khi gửi chúng trả lại dưới dạng kết quả cho truy vấn. Nếu một trường được đặt thành Lưu trữ, RavenDB sẽ lưu trữ các giá trị trường gốc trong chính chỉ mục và tải chúng từ đó khi tải kết quả truy vấn, thay vì tải dữ liệu từ tài liệu gốc.

Bằng cách lưu giá trị được tính toán vào trường Đã lưu, bạn có thể nhận các giá trị để hiển thị ngay cả khi chúng không có trong chính tài liệu. Và đó là ngoài việc các tài liệu có thể được tìm kiếm theo các trường đó.

- Full-text Search, tùy chọn khác nhau có thể được đặt trên các trường chuỗi để kích hoạt tìm kiếm toàn văn bản và chúng kích hoạt các tính năng thú vị như Suggestion hoặc Highlighting các thuật ngữ trong kết quả. Điều này bao gồm các cấu hình Indexing, Analyzer, Term Vector và Suggestion mà người ta có thể đặt cho trường đã chọn.
- Highlighting: Trong các tùy chọn nâng cao, Indexing cần được đặt thành 'Search' và Term Vector được đặt thành 'WithPositionsAndOffsets'.
- Suggestion: Cho phép bạn truy vấn xem người dùng có thể muốn hỏi về điều gì (tức là lỗi chính tả).
- Spatial: Được dùng để tìm kiếm về mặt không gian với dữ liệu địa lý. RavenDB sử dụng các thuộc tính kinh độ và vĩ độ của tài liệu, đồng thời tạo trường không gian 'Tọa độ' mà sau đó các truy vấn không gian có thể sử dụng.
 - RavenDB hỗ trợ cả loại không gian Địa lý và Descartes.
 - Chiến lược lập chỉ mục: xác định định dạng của các giá trị thuật ngữ được lập chỉ mục.
 - * Bounding box
 - * Geohash prefix tree
 - * Quad prefix tree

Các tùy chọn nâng cao:



Hình 3.11: Các tùy chọn nâng cao trong Studio

- Term Vector: tính năng này gợi ý các tài liệu tương tự với tài liệu đã chọn, dựa trên các tùy chọn như 'WithPositionsAndOffsets', 'WithPositions',... Term Vector cho một đoạn văn bản sẽ chứa danh sách tất cả các từ duy nhất và tần suất chúng xuất hiện.
- Indexing: Xác định Analyzer nào có thể được sử dụng.
 - Exact: Analyzer: 'Keyword'
 - * Text không được chia thành các tokens.
 - * Toàn bộ giá trị của trường được xử lý dưới dạng một token.
 - Default: Analyzer: 'LowerCase Keyword'
 - * Văn bản không được chia thành các tokens.
 - * Văn bản được chuyển đổi thành chữ thường và các kết quả khớp không phân biệt chữ hoa chữ thường.
 - Search: Chọn Analyzer. 'StandardAnalyzer' sẽ được sử dụng theo mặc định.
 - Analyzer: Đây là Analyzer đang được sử dụng.

3.2.7 Một số chỉ mục khác

Các chỉ mục khác được đề cập dưới đây dựa trên cơ sở của những chỉ mục đã nói trước đó.

Multi Collection Indexes

Gồm các loại chỉ mục *Multi Map Indexes* và *Multi Map-Reduce Indexes*.

Dữ liệu từ một số collection có thể được lập chỉ mục (mỗi collection trong một Map khác nhau) và kết quả được hợp nhất trong một chỉ mục duy nhất. Yêu cầu duy nhất là tất cả các định nghĩa Map phải có hình dạng đầu ra giống nhau.

Fanout Indexes

Chỉ mục cho ra nhiều Index Entries cho mỗi document. Chẳng hạn, lập chỉ mục Orders/ByProduct. Vì 1 Order có nhiều Product trong mỗi Line, do đó khi lập chỉ mục theo Product sẽ cho ra nhiều Index Entries cho các Product khác nhau với cùng 1 OrderId.

3.2.8 Index state and prioritization

Ưu tiên chỉ mục đóng một vai trò quan trọng trong việc giảm căng thẳng cho máy chủ khi có dữ liệu mới và giảm thiểu khả năng nhận được kết quả cũ cho các truy vấn. Các mức độ ưu tiên của chỉ mục được RavenDB quản lý tự động đối với tất cả các chỉ mục tự động. Bằng cách tính toán các chỉ mục được sử dụng thường xuyên hơn trước tiên, bạn có thể tăng tốc chúng nhanh hơn và giảm thời gian để các bản cập nhật được thực hiện cho máy chủ xuất hiện trong kết quả cho các truy vấn được thực hiện đối với các chỉ mục đó. Ngoài ra, cũng có thể can thiệp và đặt mức độ ưu tiên trên chỉ mục theo cách thủ công.

- High: Xử lý yêu cầu và lập chỉ mục sẽ có cùng mức độ ưu tiên ở cấp hệ điều hành.
- Normal: Các yêu cầu tới cơ sở dữ liệu vẫn được ưu tiên hơn quá trình lập chỉ mục
- Low: Các chỉ mục sẽ chỉ chạy khi có đủ dung lượng cho chúng, khi hệ thống không bận rộn với các tác vụ có mức độ ưu tiên cao hơn.

Trạng thái chỉ mục:

- Normal: Hoạt động chỉ mục bình thường. Bất cứ khi nào có dữ liệu mới, nó sẽ được xử lý ngay lập tức bởi tất cả các chỉ mục tự động ở trạng thái Bình thường.
- Disabled: Các chỉ mục được đặt là Disabled sẽ hoàn toàn không tham gia lập chỉ mục và thực tế sẽ bị tắt. Đây là một tính năng hữu ích khi bạn mong muốn có nhiều dữ liệu được đưa vào, chẳng hạn như trong quá trình nhập dữ liệu và bạn không muốn làm chậm máy chủ hoặc chính quá trình nhập.
- Paused: Cũng giống như Disabled, dữ liệu mới không được lập chỉ mục và các truy vấn sẽ cũ vì dữ liệu mới không được lập chỉ mục. Tuy nhiên, nó sẽ tự động tiếp tục khi khởi động lại máy chủ hoặc tải lại cơ sở dữ liệu.
- Idle: Auto Index trả nên nhàn rỗi khi chênh lệch thời gian giữa lần truy vấn cuối cùng và thời gian gần đây nhất mà cơ sở dữ liệu được truy vấn (với bất kỳ chỉ mục nào khác) lớn hơn ngưỡng có thể định cấu hình (theo mặc định là 30 phút).

Điều này được thực hiện để tránh đánh dấu các chỉ mục là Disabled đối với cơ sở dữ liệu ngoại tuyến trong một thời gian dài - không có dữ liệu mới để lập chỉ mục và không được truy vấn nói chung, cũng như đối với các cơ sở dữ liệu vừa được khôi phục từ ảnh chụp nhanh hoặc bản sao lưu.

Auto Index sẽ tiếp tục công việc của nó và quay trở lại trạng thái 'Normal' khi có truy vấn mới hoặc khi đặt lại chỉ mục. Nếu không được tiếp tục, chỉ mục tự động không hoạt động sẽ bị máy chủ xóa sau một khoảng thời gian có thể định cấu hình (mặc định là 72 giờ).



- Abandoned: Một chỉ mục không được truy vấn trong 72 giờ sẽ được đánh dấu là Abandoned, khi đó chỉ xử lý các tài liệu mới sau khi máy chủ không hoạt động trong khoảng 3 giờ. Các chỉ mục mới tạo được đánh dấu là Abandoned sẽ được coi là tạm thời và tự động bị xóa khi chúng đạt đến trạng thái này.
- Error: Chức năng lập chỉ mục không đúng định dạng hoặc dữ liệu tài liệu bị thiếu/hỗn sê dẫn đến lỗi lập chỉ mục.
- Faulty: Một chỉ mục sẽ bị 'Faulty' nếu các tệp dữ liệu của nó bị hỏng hoặc không thể truy cập được.



3.3 Query processing và optimization

Các truy vấn trong RavenDB sử dụng ngôn ngữ giống SQL có tên là **RavenDB Query Language** hay **RQL**.

3.3.1 Query optimizer - Quản lý indexing trong RavenDB

Khi một truy vấn được xử lý, điều đầu tiên xảy ra là nó sẽ được trình tối ưu hóa truy vấn phân tích. Vai trò của trình tối ưu hóa truy vấn là **xác định những chỉ mục** nào sẽ được sử dụng cho truy vấn cụ thể này.

Với RavenDB, có hai loại truy vấn:

- Truy vấn động: chẳng hạn như `from Order where ...`, điều này mang lại cho trình tối ưu hóa truy vấn toàn quyền tự do liên quan đến chỉ mục mà truy vấn đó sẽ sử dụng.
- Truy vấn chỉ định một chỉ mục cụ thể sẽ được sử dụng, chẳng hạn như `from index "Orders/ByCompany" where ...`, hướng dẫn RavenDB sử dụng chỉ mục `Orders/ByCompany` cho câu truy vấn này.

Các truy vấn trong RavenDB luôn sử dụng chỉ mục. Trong khi với các cơ sở dữ liệu khác, trình tối ưu hóa truy vấn có thể không tìm thấy chỉ mục phù hợp và quay lại truy vấn bằng cách quét toàn bộ, nhưng RavenDB thì không hỗ trợ quét toàn bộ. Các truy vấn trong RavenDB rất nhanh và chúng sẽ luôn sử dụng một chỉ mục. Sử dụng tính năng quét toàn bộ là lựa chọn tuyệt vời khi kích thước dữ liệu rất nhỏ nhưng khi kích thước dữ liệu bắt đầu tăng lên, chúng ta sẽ phải đổi mặt với thời gian truy vấn ngày càng tăng. Ngược lại, các truy vấn RavenDB luôn sử dụng một chỉ mục và có thể trả về kết quả với cùng tốc độ bất kể kích thước của dữ liệu.

Nếu trình tối ưu hóa truy vấn không thể tìm thấy chỉ mục có thể đáp ứng một truy vấn, thay vì quét tất cả các tài liệu, kiểm tra lần lượt từng tài liệu và đưa nó vào truy vấn hoặc loại bỏ nó do không khớp, trình tối ưu hóa truy vấn sẽ tạo một chỉ mục cho truy vấn này một cách nhanh chóng.

Đối với các cơ sở dữ liệu quan hệ, việc tạo chỉ mục cần được xem xét cẩn thận nếu không muốn dẫn đến những kết quả không đáng có. Đối với RavenDB, các chỉ mục sẽ không khóa dữ liệu và chúng được thiết kế để không tiêu tốn toàn bộ tài nguyên hệ thống khi chúng đang chạy. Việc thêm một chỉ mục là một sự kiện thường lệ nên trình tối ưu hóa truy vấn sẽ tự chạy nếu cần.

Bây giờ, việc tạo chỉ mục cho mỗi truy vấn sẽ dẫn đến khá nhiều chỉ mục trong cơ sở dữ liệu, đây vẫn không phải là một ý tưởng hay. Thay vào đó, khi nhận được một truy vấn, trình tối ưu hóa sẽ phân tích truy vấn đó và xem chỉ mục nào có thể trả lời truy vấn đó. Nếu không có, trình tối ưu hóa truy vấn sẽ tạo một chỉ mục có thể trả lời truy vấn này và tất cả các truy vấn trước đó trên collection đó.

Lập chỉ mục trong RavenDB là một thao tác ở chế độ nền, có nghĩa là truy vấn mới sẽ đợi chỉ mục hoàn tất việc lập chỉ mục (hoặc hết thời gian chờ). Nhưng đồng thời, các truy vấn có thể được trả lời bằng cách sử dụng các chỉ mục hiện có sẽ tiến hành bình thường bằng cách sử dụng các chỉ mục này. Khi chỉ mục mới đã bắt kịp, RavenDB sẽ dọn sạch tất cả các chỉ mục cũ hiện được thay thế bằng chỉ mục mới.

Nói tóm lại, theo thời gian, trình tối ưu hóa truy vấn sẽ phân tích tập hợp các truy vấn bạn thực hiện đối với cơ sở dữ liệu của mình và sẽ tạo ra bộ chỉ mục tối ưu để trả lời các truy vấn đó. Những thay đổi trong truy vấn của bạn cũng sẽ gây ra thay đổi về chỉ mục trên cơ sở dữ liệu của bạn khi nó điều chỉnh theo các yêu cầu mới.

3.3.2 RavenDB Query Language - RQL

Đầu tiên cần hiểu là RavenDB sẽ xử lý từng trường trong truy vấn một cách riêng biệt. Lý do RavenDB xử lý riêng từng trường là vì nó lưu trữ dữ liệu được lập chỉ mục cho từng trường một cách độc lập. Điều này cho phép chúng ta tự do hơn rất nhiều trong thời gian truy vấn, nhưng lại phải làm



thêm một chút công việc.

Các chỉ mục kết hợp tất cả các trường lại với nhau thành một khóa duy nhất và sau đó cho phép tìm kiếm trên kết quả sẽ rất hiệu quả trong việc trả lời loại truy vấn cụ thể đó, nhưng chúng thực sự không thể được sử dụng cho bất kỳ mục đích nào khác. Với RavenDB, chỉ mục được lập cho từng trường một cách độc lập và hợp nhất các kết quả tại thời điểm truy vấn. Điều đó có nghĩa là các chỉ mục có thể được sử dụng theo cách linh hoạt hơn và chúng có thể trả lời phạm vi truy vấn rộng hơn nhiều với chi phí nhỏ cho công việc bổ sung để hợp nhất chúng tại thời điểm truy vấn.

Kết quả cuối cùng của một câu truy vấn là một chỉ mục duy nhất trong RavenDB có thể được sử dụng cho nhiều loại truy vấn hơn so với một chỉ mục tương tự trong cơ sở dữ liệu quan hệ. Đây là chi phí thực hiện các thao tác tập hợp trên các truy vấn có nhiều mệnh đề. Vì các thao tác thiết lập khá rẻ và đã được tối ưu hóa cẩn thận nên đó là một sự đánh đổi khá tốt để thực hiện.

Toán tử trong câu truy vấn

Operation	Operators/Methods
Equality	=, ==, !=, <>, IN, ALL IN
Range queries	>, <, >=, <=, BETWEEN
Text search	Exact, StartsWith, EndsWith, Search
Aggregation	Count, Sum, Avg
Spatial	spatial.Contains, spatial.Within, spatial.Intersects
Other	Exists, Lucene, Boost

Equality operators

Toán tử đầu tiên và rõ ràng nhất là so sánh đẳng thức ('=' hoặc '=='). Đây là những thứ dễ tìm thấy nhất vì chỉ có thể kiểm tra chỉ mục để tìm giá trị mà so sánh.

Điều quan trọng cần lưu ý là RavenDB chỉ cho phép so sánh các trường với các giá trị hoặc tham số. Loại truy vấn này phù hợp: `where FirstName = 'Andrew'` hoặc như thế này: `where FirstName = $name`. Tuy nhiên, điều này không được phép: `where FirstName = LastName`.

Đối với các truy vấn bắt bình đẳng, vì RavenDB sử dụng tập hợp các thao tác để tính toán kết quả truy vấn. Một truy vấn chẳng hạn như ở đâu `where FirstName != 'Andrew'` thực sự được dịch sang: `where exists(FirstName) and not FirstName = 'Andrew'`. Nói cách khác, có thể diễn giải câu truy vấn trên thành **Tìm tất cả tài liệu có trường FirstName và loại trừ tất cả tài liệu trong đó FirstName được đặt thành 'Andrew'**.

Ngoài ra còn có IN, có thể được sử dụng trong các truy vấn chẳng hạn như `where Address.City IN ('London', 'New York')` — một cách viết ngắn hơn `where Address.City = 'London' or Address.City = 'New York'`. Tuy nhiên, IN cũng cho phép bạn gửi một đối số mảng và viết truy vấn đơn giản như `where Address.City IN ($cities)`. Mặt khác, ALL IN truy vấn sẽ khớp với `where Address.City = 'London' and Address.City = 'New York'`. Nói cách khác, nó sẽ sử dụng and thay vì or.

Rất là vô lý khi một biến không thể mang đồng thời 2 giá trị. Nhưng một mảng thì chắc chắn là có thể. Hãy xem xét tài liệu, với một loạt các vùng lãnh thổ. Chúng ta có thể sử dụng ALL IN trong truy vấn của mình để tìm tất cả các khu vực có nhiều lãnh thổ trong đó:

```
from Regions where Territories[] .Name ALL IN ('Wilton', 'Neward')
```

Truy vấn này hiển thị hai tính năng mới:

- Dầu tiên, chúng ta có ALL IN, cho thấy cách chúng ta có thể khớp nhiều giá trị với một mảng. Cách sử dụng phổ biến của tính năng này là lọc tài liệu theo thẻ (tags). Chúng ta có thể chọn những thẻ quan tâm và sử dụng ALL IN để tìm tất cả các tài liệu khớp với các thẻ được yêu cầu.



- Tính năng mới thứ hai là việc sử dụng đường dẫn `Territories[]`. `Name` và đặc biệt là việc sử dụng `[]` trong đường dẫn. Trong RQL, việc sử dụng hậu tố `[]` trong thuộc tính cho biết đây là một mảng và phần còn lại của biểu thức được lồng vào các giá trị của mảng. Điều này hữu ích cả trong mệnh đề `Where` và khi thực hiện các phép chiếu bằng cách sử dụng `select`.

Range queries

Trong khi các toán tử như `<`, `>`, `<=`, `>=` khá dễ hiểu và tương tự, khi cung cấp các cơ chế truy vấn trên 1 giới hạn cụ thể. Trong khi đó `BETWEEN` cung cấp cơ chế tốt hơn để thực sự truy vấn trên một phạm vi cụ thể, bao gồm cả giới hạn ở 2 đầu.

Xem xét ví dụ sau: `from Employees where HiredAt.Year BETWEEN 1992 AND 1994`. Kết quả của truy vấn sẽ bao gồm các nhân viên được thuê vào năm 1992, 1993 và 1994.

Chúng ta cũng có thể viết truy vấn tương tự với chuỗi khớp (string matching): `from Employees where HiredAt BETWEEN '1992' AND '1995'`. Truy vấn này cũng sẽ khớp với tất cả các nhân viên được thuê vào năm 1992, 1993 và 1994. Điều này xảy ra là bởi vì định dạng ngày thực tế được RavenDB sử dụng là ISO 8601, vì vậy về mặt kỹ thuật, truy vấn này sẽ trông giống như sau: `HiredAt BETWEEN '1992-01-01T00:00:00.0000000' AND '1995-01-01T00:00:00.0000000'`. Trong thực tế, RavenDB coi các truy vấn như vậy là các thao tác chuỗi và cho phép chúng ta thực hiện thao tác `BETWEEN` chỉ bằng tiền tố.

Điều này là do cách RavenDB xử lý các truy vấn phạm vi. Đối với các giá trị không phải là số, truy vấn phạm vi sử dụng so sánh từ vựng, nghĩa là chỉ cần xác định tiền tố là đủ để chúng ta nhận được kết quả cần thiết. Đó là lý do tại sao RavenDB sử dụng ngày ISO 8601. Chúng sắp xếp theo từ vựng, giúp mọi việc trở nên dễ dàng hơn tại thời điểm truy vấn.

Tất nhiên, đối với các giá trị số. Khi RavenDB lập chỉ mục một giá trị số, nó thực sự sẽ lập chỉ mục giá trị đó nhiều lần: một lần dưới dạng chuỗi, cho phép nó tham gia so sánh từ vựng và một lần dưới dạng giá trị số. Trên thực tế, nó thậm chí còn phức tạp hơn thế. Vẫn đê là khi chúng ta làm việc với máy tính, việc xác định một số thực sự hơi phức tạp.

RavenDB hỗ trợ hai loại số: số nguyên 64 bit (64-bit integers) và dấu phẩy động có độ chính xác kép IEEE 754 (IEEE 754 double-precision floating-points). Khi RavenDB lập chỉ mục một trường số, việc lập chỉ mục trường đó sẽ diễn ra ba lần: một lần dưới dạng chuỗi (string), một lần dưới dạng double và một lần dưới dạng int64. Và nó cho phép truy vấn tất cả chúng mà không thực sự quan tâm đến những gì sử dụng để tìm kết quả.

Full text searching

Giữa 2 câu truy vấn `from Employees where FirstName = 'Andrew'` và `from Employees where FirstName = 'ANDREW'` đều cho ta được kết quả mong muốn như nhau, đều tìm các nhân viên có tên (firstname) là Andrew; bởi vì trong RavenDB mặc định sử dụng các câu truy vấn khớp với không phân biệt chữ hoa chữ thường (**case-insensitive**).

Mặt khác, có thể viết truy vấn `from Employees where exact.FirstName = 'Andrew'` và chỉ tìm thấy các kết quả khớp với giá trị và cách viết hoa được sử dụng. Trong phạm vi `exact`, tất cả các so sánh đều sử dụng kết quả khớp phân biệt chữ hoa chữ thường (**case-sensitive**).

Theo mặc định, các truy vấn trong RavenDB không phân biệt chữ hoa chữ thường, điều này giúp ích rất nhiều. Nhưng điều gì sẽ xảy ra khi chúng ta cần nhiều hơn như thế, có thể sử dụng `StartsWith` và `EndsWith` để giải quyết các truy vấn. Hãy xem xét truy vấn sau:

```
from Employees where StartsWith(FirstName, 'An').
```

Điều này sẽ tìm thấy tất cả các nhân viên có tên bắt đầu bằng 'An'. Điều tương tự có thể được thực hiện với `Where EndsWith(LastName, 'er')`.



Lưu ý rằng các truy vấn sử dụng StartsWith có thể sử dụng chỉ mục một cách hiệu quả để thực hiện tìm kiếm tiền tố, nhưng EndsWith là thứ sẽ khiến RavenDB thực hiện quét toàn bộ chỉ mục. Vì vậy, EndsWith không được khuyến khích sử dụng thông thường. Nếu thực sự cần tính năng này, có thể sử dụng chỉ mục tĩnh để lập chỉ mục ngược lại trường đang tìm kiếm và sử dụng StartsWith, việc này sẽ nhanh hơn nhiều.

Cuối cùng là khả năng thực hiện tìm kiếm toàn văn trên dữ liệu. Tìm kiếm toàn văn cho phép tìm kiếm một thuật ngữ (hoặc các thuật ngữ) cụ thể trong một bộ tài liệu và tìm kết quả mà không cần phải khớp chính xác. Với câu truy vấn:

```
from Companies where search(Name, "Stop")
```

Câu truy vấn này sẽ thực hiện tìm kiếm các công ty có tên chứa "stop". Thay vì lập chỉ mục trường Name dưới dạng một giá trị duy nhất, RavenDB sẽ chia nó thành các token riêng biệt. Điều này có nghĩa là có thể tìm kiếm từng từ riêng lẻ bên trong các thuật ngữ; ta không tìm kiếm toàn bộ trường mà tìm kiếm các token được lập chỉ mục và từ đó, ta sẽ tìm thấy các tài liệu phù hợp.

Tìm kiếm toàn văn trong RavenDB cũng cho phép thực hiện không chỉ tìm kiếm trên chuỗi mà còn ở trên 1 đối tượng. Xem xét truy vấn sau:

```
from Companies where search(Address, "London Sweden").
```

Thuộc tính Address trên tài liệu Companies không phải là một chuỗi đơn giản; nó thực sự là một đối tượng lồng nhau. Nhưng RavenDB không gặp vấn đề gì khi lập chỉ mục toàn bộ đối tượng. Kết quả của truy vấn này bao gồm các công ty cư trú tại thành phố Luân Đôn (city - London) hoặc quốc gia Thụy Điển (country - Sweden). Tùy chọn mạnh mẽ này cho phép bạn tìm kiếm trên các đối tượng phức tạp một cách dễ dàng.

Spatial queries

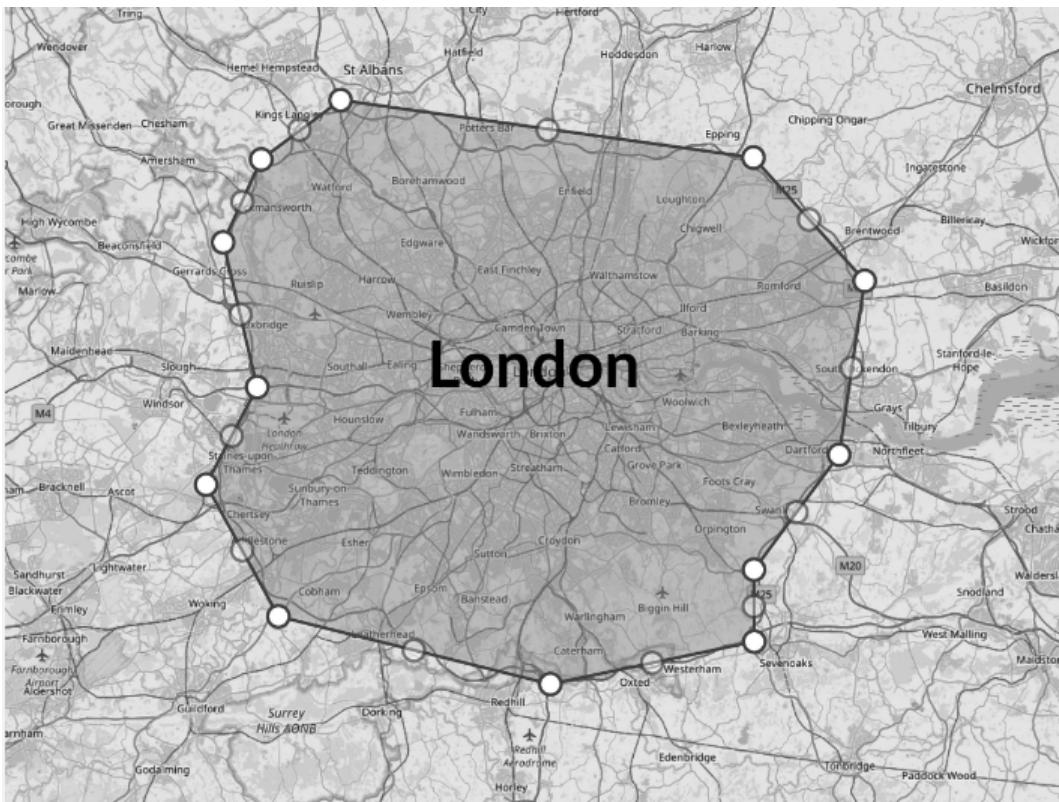
Tìm kiếm không gian cho phép tìm kiếm bằng cách sử dụng dữ liệu địa lý. Tất nhiên, các truy vấn không gian yêu cầu phải có dữ liệu không gian, chẳng hạn như vị trí kinh độ/vĩ độ trên địa cầu. Các truy vấn không gian được thể hiện bằng các phép toán không gian cơ bản sau: `spatial.within()`, `spatial.intersects()`, `spatial.contains()` and `spatial.disjoint()`. Xét câu truy vấn sau:

```
from Employees
where spatial.within(
    spatial.point(Address.Location.Latitude, Address.Location.Longitude),
    spatial.circle(20, 47.448, -122.309, 'kilometers')
).
```

Câu truy vấn này sẽ tìm kiếm những nhân viên có địa chỉ nằm trong phạm vi 20km của điểm có hoành độ, tung độ lần lượt là 47.448 và -122.309. Tuy nhiên, trong thực tế, có thể có nhiều yêu cầu còn phức tạp hơn thế này. Nếu ta cần tìm các nhân viên có địa chỉ không phải ở trong 1 hình tròn mà ở trong 1 đa giác (cụ thể là 1 vùng lãnh thổ, thành phố, quốc gia,...), như trong ví dụ dưới đây.

Để có thể tìm được các nhân viên có địa chỉ ở trong khu vực London (được xấp xỉ bằng 1 đa giác) thì ta thực hiện câu truy vấn như sau:

```
from Employees
where spatial.within(
    spatial.point(Address.Location.Latitude, Address.Location.Longitude),
    spatial.wkt("POLYGON((
        0.1483154296875 51.67881439742299,
        0.2911376953125 51.579928527080114,
```



0.2581787109375 51.439756376733676,
0.1483154296875 51.347212267024645,
0.1483154296875 51.288847685894844,
-0.1153564453125 51.25448088572911,
-0.4669189453125 51.3094554292733,
-0.560302734375 51.41578143396663,
-0.494384765625 51.494509016952534,
-0.538330078125 51.61064031418932,
-0.490264892578125 51.677111294565,
-0.38726806640625 51.72477396651261
)))")

Aggregation queries

Việc tổng hợp trong RavenDB có thể được sử dụng trực tiếp từ RQL bằng cách sử dụng **group by** hoặc bằng cách xây dựng các chỉ mục MapReduce. Tuy nhiên, trong cả hai trường hợp, việc triển khai cơ bản đều giống nhau. Việc tổng hợp trong RavenDB xảy ra trong quá trình lập chỉ mục chứ không phải trong khi truy vấn và do đó nó rẻ hơn nhiều so với các truy vấn tổng hợp trong các cơ sở dữ liệu khác.

Hãy bắt đầu bằng cách xem xét truy vấn tổng hợp đơn giản sau:

```
from Orders
group by Company
where count() > 25
order by count() as long desc
select count(), Company
```

Truy vấn này khá đơn giản: liệt kê các công ty có nhiều đơn đặt hàng nhất theo thứ hạng giảm dần nếu họ có tổng cộng hơn 25 đơn đặt hàng. Đơn giản và rõ ràng. Nhưng có rất nhiều điều đang diễn ra đằng sau hậu trường. Như thường lệ, trình tối ưu hóa truy vấn sẽ tạo một chỉ mục để chúng ta trả lời truy vấn này. Chỉ mục này sẽ phục vụ tất cả các truy vấn về Đơn hàng được nhóm theo Công ty.

Điều thú vị là chỉ mục sẽ xử lý việc tổng hợp thực tế cho chúng ta. Nói cách khác, trong quá trình lập chỉ mục, chúng ta sẽ thực hiện việc nhóm và ghi các kết quả đã được tính toán vào chỉ mục. Khi truy vấn, chúng tôi không cần thực hiện thêm bất kỳ công việc nào — chỉ tra cứu tất cả các công ty trong chỉ



mục có hơn 25 đơn hàng và trả lại ngay lập tức. Khi có đơn hàng mới, chỉ mục sẽ không cần tính toán lại toàn bộ tập hợp từ đầu mà chỉ có thể cập nhật các chi tiết liên quan.

Bất kể chúng ta đang sử dụng cái gì — một nhóm theo truy vấn của bạn hay chỉ mục MapReduce — ý tưởng rằng các truy vấn tổng hợp rẻ có một số ý nghĩa thực sự thú vị đối với hệ thống. Khi sử dụng truy vấn tổng hợp, không cần phải xử lý bộ nhớ đệm hoặc có bộ lập lịch để tính toán lại kết quả, tất cả việc đó đều được thực hiện.

Trong cơ sở dữ liệu quan hệ, loại truy vấn này thực sự khá tốn kém. Ngay cả khi giả sử chúng ta có một chỉ mục trên trường Công ty, cơ sở dữ liệu quan hệ sẽ cần xem qua tất cả các kết quả và đếm chúng, lọc những kết quả không có đủ, sắp xếp và sắp xếp chúng. Nếu số lượng đơn hàng cao, điều đó có nghĩa là cơ sở dữ liệu quan hệ cần đọc qua tất cả các đơn hàng (hoặc ít nhất là toàn bộ chỉ mục trên cột Công ty).

Mặt khác, RavenDB đã thực hiện công việc này trong quá trình lập chỉ mục, vì vậy khi bạn thực hiện truy vấn, RavenDB chỉ lấy tất cả các công ty có hơn 25 đơn đặt hàng, sắp xếp chúng và trả về kết quả.

3.4 Transaction processing

Session (Phiên) được lấy từ Document Store, là một Đơn vị công việc (Unit of work) đại diện cho một transaction nghiệp vụ đơn lẻ trên một cơ sở dữ liệu cụ thể. Các thao tác CRUD tài liệu cơ bản và Truy vấn tài liệu có sẵn thông qua Phiên. Các tùy chọn nâng cao hơn có sẵn bằng cách sử dụng các thao tác Phiên nâng cao. Phiên theo dõi tất cả các thay đổi được thực hiện đối với tất cả các thực thể mà nó đã tải, lưu trữ hoặc truy vấn, và chỉ lưu lại trên máy chủ những gì cần thiết khi SaveChanges() được gọi.

3.4.1 Unit of work pattern

Theo dõi thay đổi - Tracking changes

- Sử dụng Phiên, thực hiện các thao tác cần thiết trên tài liệu, ví dụ: tạo một tài liệu mới, sửa đổi tài liệu hiện có, truy vấn tài liệu, v.v.
- Bất kỳ thao tác nào như vậy đều 'loads' tài liệu dưới dạng một thực thể vào Phiên, và nó được thêm vào bản đồ thực thể (entities map) của Phiên.
- Phiên theo dõi tất cả các thay đổi được thực hiện đối với tất cả các thực thể được lưu trữ trong bản đồ nội bộ của nó.
- Chúng ta không cần phải theo dõi thủ công các thay đổi đối với các thực thể này và quyết định những gì cần lưu và những gì không. Phiên sẽ làm điều đó.
- Tất cả những thay đổi được theo dõi này chỉ được kết hợp và lưu giữ trong cơ sở dữ liệu khi gọi SaveChanges().

3.4.2 Identity map pattern

Phiên triển khai Identity Map Pattern.

- Lệnh gọi Load() đầu tiên đến máy chủ và lấy tài liệu từ cơ sở dữ liệu. Sau đó, tài liệu được lưu trữ dưới dạng thực thể trong bản đồ thực thể của Phiên.
- Tất cả các lệnh gọi Load() tiếp theo tới cùng một tài liệu sẽ chỉ truy xuất thực thể từ Phiên - không có cuộc gọi bổ sung nào đến máy chủ được thực hiện.

3.4.3 Batching & Transactions

3.4.3.1 Batching

- Các cuộc gọi từ xa đến máy chủ qua mạng là một trong những hoạt động tốn kém nhất mà một ứng dụng thực hiện.
- Phiên này tối ưu hóa điều này bằng cách gộp tất cả các thao tác ghi mà nó đã theo dõi vào lệnh gọi SaveChanges().
- Khi gọi SaveChanges(), phiên sẽ kiểm tra trạng thái của nó để biết tất cả các thay đổi đã thực hiện cần được lưu trong cơ sở dữ liệu, và kết hợp chúng thành một lô (batch) duy nhất được gửi đến máy chủ dưới dạng một cuộc gọi từ xa.

3.4.3.2 Transactions

- Các hoạt động theo đợt được gửi trong SaveChanges() sẽ hoàn tất transaction. Nói cách khác, tất cả các thay đổi đều được lưu dưới dạng Single Atomic Transaction hoặc không có thay đổi nào trong số đó. Vì vậy, khi SaveChanges() trả về thành công, đảm bảo rằng tất cả các thay đổi sẽ được lưu vào cơ sở dữ liệu.
- SaveChanges() là lần duy nhất khi máy khách RavenDB gửi các bản cập nhật đến máy chủ từ Phiên, vì vậy số lượng cuộc gọi mạng giảm đi.

3.4.3.3 Transaction mode

Chế độ transaction của phiến có thể được đặt thành:

- Nút đơn (Single Node) - transaction được thực hiện trên một nút cụ thể và sau đó được sao chép
- Toàn cụm (Cluster Wide) - transaction được đăng ký để thực hiện trên tất cả các nút theo kiểu atomic

3.4.4 Single-Node transaction

- Transaction ở một nút được coi là thành công khi được thực hiện thành công trên nút mà máy khách đang liên lạc. Dữ liệu có sẵn ngay lập tức trên nút đó và cuối cùng nó sẽ nhất quán trên tất cả các nút cơ sở dữ liệu khác khi quá trình sao chép diễn ra ngay sau đó.
- Mọi hành động đều có sẵn ngoại trừ PUT/DELETE một mục trao đổi so sánh. Và không có Atomic-Guard nào được tạo bởi máy chủ.
- Xung đột có thể xảy ra khi hai transaction đồng thời sửa đổi cùng một tài liệu trên các nút khác nhau cùng một lúc. Chúng được giải quyết theo cài đặt xung đột đã xác định, bằng cách sử dụng phiên bản mới nhất (mặc định) hoặc bằng cách làm theo tập lệnh giải quyết xung đột. Các bản sửa đổi được tạo cho các tài liệu xung đột để có thể phục hồi bất kỳ tài liệu nào.
- Transaction một nút được coi là nhanh hơn và ít tốn kém hơn vì không cần có sự đồng thuận của cụm để thực hiện.

3.4.5 Cluster-Wide transaction

- Các transaction trên toàn cụm là các transaction **ACID hoàn toàn (fully ACID)** trên tất cả các nút nhóm cơ sở dữ liệu. Được triển khai bằng thuật toán Raft, cụm trước tiên phải đạt được sự đồng thuận. Khi phần lớn các nút (node) đã phê duyệt transaction, transaction được đăng ký để thực hiện trong hàng đợi transaction của tất cả các nút theo kiểu nguyên tử (atomic).
- Transaction sẽ thành công trên tất cả các nút hoặc được khôi phục. Transaction chỉ được coi là thành công khi được đăng ký thành công trên tất cả các nút nhóm cơ sở dữ liệu. Sau khi được thực thi trên tất cả các nút, dữ liệu sẽ nhất quán và có sẵn trên tất cả các nút. Việc không đăng ký transaction trên bất kỳ nút nào sẽ khiến transaction bị khôi phục trên tất cả các nút và các thay đổi sẽ không được áp dụng.
- Các hành động duy nhất có sẵn là: PUT/DELETE một tài liệu và PUT/DELETE một mục trao đổi so sánh
- Để tránh việc sửa đổi các tài liệu đồng thời, máy chủ tạo Atomic-Guards sẽ được liên kết với các tài liệu. Atomic-Guard sẽ được tạo khi: Một tài liệu mới được tạo hoặc Sửa đổi tài liệu hiện có chưa có Atomic-Guard.
- Các transaction trên toàn cụm không có xung đột (conflict-free).
- Transaction trên toàn cụm được coi là đắt hơn và kém hiệu quả hơn do cần có sự đồng thuận của cụm trước khi thực hiện.

3.5 Concurrency control

3.5.1 Change vector

Các change vector là cách triển khai của khái niệm vector clock trong RavenDB - cho ta thấy được thứ tự một phần đối với việc sửa đổi tài liệu trong cụm RavenDB. Các change vector được cập nhật mỗi khi tài liệu được sửa đổi theo bất kỳ cách nào.

Một change vector được xây dựng từ các mục, mỗi mục một cơ sở dữ liệu:

[A:1-0tIXNUeUckSe73dUR6rjrA, B:7-kSXfVRAkKEmfZpyfkD+Zw]. Change vector này được xây dựng từ hai mục, A:1-0tIXNUeUckSe73dUR6rjrA và B:7-kSXfVRAkKEmfZpyfkD+Zw.

Mỗi mục nhập có cấu trúc sau Node Tag: ETag-Database ID, vì vậy A:1-0tIXNUeUckSe73dUR6rjrA có nghĩa là tài liệu đã được sửa đổi trên nút A, ETag cục bộ của nó là 1 và ID cơ sở dữ liệu là 0tIXNUeUckSe73dUR6rjrA. Các mục nhập tích lũy khi số lượng phiên bản cơ sở dữ liệu trong nhóm cơ sở dữ liệu tăng lên. Nếu một phiên bản không còn được sử dụng nữa thì mục nhập của nó có thể bị xóa khỏi change vector để tiết kiệm dung lượng bằng cách sử dụng `UpdateUnusedDatabasesOperation`.

ETag là một giá trị nằm trong phạm vi của một phiên bản cơ sở dữ liệu trên một nút cụ thể và được đảm bảo luôn tăng. Nó được sử dụng nội bộ cho nhiều mục đích và là thứ tự sắp xếp tự nhiên cho nhiều hoạt động (lập chỉ mục, sao chép, ETL, v.v.). ID cơ sở dữ liệu được sử dụng trong trường hợp thẻ nút không phải là duy nhất, điều này có thể xảy ra khi sử dụng bản sao bên ngoài hoặc khôi phục từ bản sao lưu.

3.5.2 Các change vector được sử dụng như thế nào để xác định thứ tự?

Cho hai change vector X và Y, chúng ta sẽ nói rằng $X \geq Y$ nếu với mỗi mục nhập của X thì giá trị của ETag lớn hơn hoặc bằng mục nhập tương ứng trong Y và Y không có mục nào mà X không có. Điều tương tự cũng xảy ra với \leq , chúng ta sẽ nói rằng $X \leq Y$ nếu với mỗi mục nhập của X thì giá trị của ETag nhỏ hơn hoặc bằng mục nhập tương ứng trong Y và X không có mục nào mà Y không có. Chúng tôi sẽ nói rằng $X \neq Y$ (xung đột) nếu X có mục nhập có giá trị ETag cao hơn Y và Y có mục nhập khác trong đó giá trị ETag của nó lớn hơn X.

3.5.3 Concurrency Control

RavenDB xác định một số quy tắc đơn giản để xác định cách xử lý các hoạt động đồng thời trên cùng một tài liệu trên toàn cụm. Nó sử dụng change vector của tài liệu. Điều này cho phép RavenDB phát hiện các vấn đề đồng thời khi ghi vào tài liệu và đưa ra một ngoại lệ khi tài liệu được sửa đổi đồng thời trên các nút khác nhau trong quá trình phân vùng mạng để ngăn ngừa hỏng dữ liệu.

Mọi tài liệu trong RavenDB đều có một change vector tương ứng. Change vector này được RavenDB cập nhật mỗi khi tài liệu được thay đổi.

Một change vector có trong metadata của tài liệu và mỗi khi tài liệu được cập nhật, máy chủ sẽ cập nhật change vector. Điều này chủ yếu được sử dụng nội bộ bên trong RavenDB cho nhiều mục đích (phát hiện xung đột, quyết định tài liệu nào mà một đăng ký cụ thể đã xem, tài liệu nào được gửi đến đích ETL, v.v.) nhưng cũng có thể rất hữu ích cho khách hàng.

Đặc biệt, change vector được đảm bảo thay đổi bất cứ khi nào tài liệu thay đổi và có thể được sử dụng như một phần của kiểm tra đồng thời lạc quan. Tất cả các sửa đổi tài liệu đều có thể chỉ định một change vector dự kiến cho một tài liệu (với một change vector trống biểu thị rằng tài liệu không tồn tại). Trong trường hợp như vậy, mọi thao tác trong transaction sẽ bị hủy bỏ và không có thay đổi nào được áp dụng đối với bất kỳ tài liệu nào được sửa đổi trong transaction.



RavenDB triển khai chiến lược đa chủ để xử lý việc ghi cơ sở dữ liệu. Điều này có nghĩa là nó sẽ không bao giờ từ chối việc ghi hợp lệ vào một tài liệu (với giả định rằng nếu bạn cố gắng ghi dữ liệu vào cơ sở dữ liệu thì có thể bạn muốn giữ nó). Hành vi này có thể dẫn đến một số trường hợp khó khăn nhất định. Đặc biệt, trong kịch bản phân vùng mạng, hai máy khách có thể giao tiếp với hai nút RavenDB và cập nhật tài liệu với khả năng kiểm tra đồng thời.

Việc kiểm tra đồng thời được thực hiện ở cấp cục bộ để đảm bảo rằng vẫn có thể xử lý việc ghi trong trường hợp phân vùng mạng hoặc tình huống lỗi một phần. Điều đó có thể có nghĩa là cả hai lần ghi vào các máy chủ riêng biệt đều sẽ thành công, ngay cả khi mỗi lần ghi chỉ định cùng một change vector ban đầu, vì mỗi máy chủ đã thực hiện kiểm tra một cách độc lập. Trong kịch bản như vậy, các change vector được tạo cho tài liệu trên mỗi máy chủ sẽ khác nhau và ngay khi sao chép giữa các nút này chạy, cơ sở dữ liệu sẽ phát hiện xung đột này và giải quyết nó theo chiến lược giải quyết xung đột của cá nhân.



3.6 Backup và Recovery

Tùy chọn sao lưu cơ bản cho RavenDB là kết xuất JSON được nén bằng tất cả các tài liệu và dữ liệu khác (chẳng hạn như tệp đính kèm) bên trong cơ sở dữ liệu. Tùy chọn sao lưu này cung cấp kích thước nhỏ nhất có thể cho dữ liệu và giúp lưu trữ các tệp sao lưu dễ dàng hơn và rẻ hơn. Mặt khác, khi cần khôi phục, RavenDB sẽ cần chèn lại và lập chỉ mục lại toàn bộ dữ liệu. Điều này có thể làm tăng thời gian cần thiết để khôi phục cơ sở dữ liệu.

Một lựa chọn thay thế cho tùy chọn sao lưu là **snapshot**. Snapshot là bản sao nhị phân của cơ sở dữ liệu và tạp chí tại một thời điểm nhất định. Giống như bản sao lưu thông thường, snapshot được nén. Bên cạnh đó, quá trình khôi phục snapshot bao gồm việc trích xuất dữ liệu và tệp nhật ký từ kho lưu trữ và khởi động cơ sở dữ liệu một cách bình thường. Ưu điểm ở đây là việc khôi phục cơ sở dữ liệu bằng snapshot nhanh hơn nhiều. Tuy nhiên, snapshot cũng thường lớn hơn nhiều so với bản sao lưu thông thường.

Trong hầu hết các trường hợp, chúng ta sẽ có cả bản sao lưu thông thường được xác định để lưu trữ lâu dài (trong đó tốc độ khôi phục không quan trọng) và bản sao lưu snapshot được ghi vào bộ nhớ có thể truy cập ngay lập tức (chẳng hạn như SAN cục bộ) để khôi phục nhanh.

Cả bản sao lưu và snapshot đều lưu bản sao đầy đủ của cơ sở dữ liệu từ các thời điểm cụ thể. Tuy nhiên, có nhiều trường hợp bạn không muốn lúc nào cũng có bản sao lưu đầy đủ. Bạn có thể chỉ muốn những thay đổi đã xảy ra kể từ lần sao lưu cuối cùng. Đây được gọi là bản sao lưu gia tăng (**an incremental backup**) và có sẵn cho cả bản sao lưu và snapshot.

Bản sao lưu gia tăng được định nghĩa là tập hợp các thay đổi đã xảy ra kể từ lần sao lưu/snapshot cuối cùng. Bất kể bạn chọn chế độ sao lưu nào, bản sao lưu gia tăng sẽ luôn sử dụng JSON được nén bằng gzipped (vì RavenDB không thực hiện incremental snapshot). Lý do cho điều này là việc áp dụng các bản sao lưu gia tăng cho snapshot thường rất nhanh và sẽ không làm tăng đáng kể thời gian cần thiết để khôi phục cơ sở dữ liệu, trong khi các snapshot gia tăng có thể rất lớn. Suy cho cùng, một trong những lý do chính khiến việc sao lưu gia tăng tồn tại ngay từ đầu là để giảm chi phí thực hiện sao lưu.

CHƯƠNG 4

ỨNG DỤNG SỬ DỤNG MICROSOFT SQL SERVER

4.1 Mô tả yêu cầu

Trang thương mại điện tử **Le Livre** kinh doanh sách thông qua hình thức bán hàng trực tuyến. Khách hàng sẽ xem hình ảnh, thông tin của sách và đặt mua sách trên website, sau đó chọn phương thức thanh toán có sẵn và tiến hành thanh toán. Cuối cùng sách sẽ được chuyển đến khách hàng đến tận nhà.

Microsoft SQL Server sẽ là hệ quản trị cơ sở dữ liệu được sử dụng cho ứng dụng này. Cơ sở dữ liệu của ứng dụng sẽ bao gồm các thông tin chính như sau:

- **Thông tin của sách (Book):** Book_ID, Book_name (Tên sách), O_Price (Giá gốc), Discount (Phần trăm khuyến mãi), Price (Giá bán), Publish_year (Năm xuất bản), Description (Mô tả chung về sách), Ratings (Điểm đánh giá trung bình, tối đa 5 sao), Thumbnail (Hình ảnh minh họa của sách), Reviews_N (số lượt đánh giá), Deleted (Xóa mềm, 1 - sách không được liệt kê trên trang web, 0 - sách được liệt kê).
- **Thông tin của nhà xuất bản (Publisher):** Publisher_ID, Publisher_name (Tên nhà xuất bản)
- **Thông tin thể loại sách (Genre):** Genre_ID, Genre_name (Tên thể loại, ví dụ: Khoa học, Ngoại ngữ).
- **Thông tin tác giả (Author):** Author_ID, Author_name (Tên tác giả).
- **Thông tin tài khoản (Account):** Account_ID, FName (Tên), LName (Họ), Email, TelephoneNum, Password, Birthday, Address, Delete (Xóa mềm). Tài khoản (Account) được thừa kế bởi Staff và Customer. Mỗi Staff (Nhân viên) sẽ được gán một nhiệm vụ riêng (Role).
- **Thông tin nhiệm vụ (Role):** Đây là bảng chứa tên nhiệm vụ của nhân viên bao gồm Role_ID và Role_name (tên nhiệm vụ).
- **Thông tin đặt hàng (Order):** Order_ID, Address (Địa chỉ), Create_date (Ngày đặt hàng), Status (Trạng thái đơn hàng), Total_price (Tổng Hóa đơn), Note (Ghi chú).
- **Thông tin chi tiết từng cuốn sách trong đơn hàng (Order_detail):** Detail_ID, Price (Giá bán), Quantity (Số lượng), Total_cost (Tổng giá = Price x Quantity).
- **Phương thức vận chuyển (Shipping_method):** Shipping_ID, Fee (Giá cho mỗi Kilometre), Shipping_name (Tên phương thức vận chuyển).
- **Phương thức thanh toán:** Payment_ID, Payment_name (Tên phương thức thanh toán).
- Một số thông tin và thuộc tính khác sẽ được mô tả chi tiết ở các đặc tả.



Khách hàng muốn mua hàng hoặc thêm hàng vào giỏ cần phải đăng nhập trước, nếu chưa có tài khoản thì bắt buộc phải tạo tài khoản. Tài khoản của các nhân viên sẽ được tạo bởi người quản lý có Role_ID là 0. Tài khoản của quản lý sẽ được tạo bởi người quản lý cơ sở dữ liệu.

4.2 Phân tích yêu cầu

4.2.1 Các kiểu thực thể mạnh và mối liên kết

- Các kiểu thực thể mạnh bao gồm:

- Book
- Author
- Publisher
- Genre
- Account
- Staff
- Role
- Customer
- Order_detail
- Order
- Shipping_method
- Payment_method

- Các kiểu mối liên kết:

- Kiểu mối liên kết Write giữa Author và Book.
- Kiểu mối liên kết Publish giữa Publisher và Book.
- Kiểu mối liên kết Belongs_to giữa Book và Genre.
- Kiểu mối liên kết Has giữa Staff và Role
- Kiểu mối liên kết Confirm giữa Customer và Order.
- Kiểu mối liên kết Has giữa Book và Order_detail.
- Kiểu mối liên kết Contain giữa Order và Order_detail.
- Kiểu mối liên kết Has giữa Order và Shipping_method.
- Kiểu mối liên kết Has giữa Order và Payment_method.

4.2.2 Các kiểu thực thể yếu và mối liên kết

- Các kiểu thực thể yếu:

- Reviews (phụ thuộc vào thực thể mạnh là: Account và Book).

- Các kiểu mối liên kết:

- Kiểu mối liên kết Comment giữa thực thể mạnh Account và thực thể yếu Reviews.
- Kiểu mối liên kết Has giữa thực thể mạnh Book và thực thể yếu Reviews.



4.2.3 Các thuộc tính và mô tả

- Thực thể **Book** có các thuộc tính Book_ID, Book_name (Tên sách), O_Price (Giá gốc), Discount (Phần trăm giảm giá), Price (Giá bán), Publish_year (Năm xuất bản), Description (Mô tả chung về sách), Ratings (Trung bình điểm đánh giá, tối đa 5 điểm), Thumbnail (Hình ảnh minh họa của sách), Reviews_N (Số lượt đánh giá), Deleted (Xóa mềm), Quantity (Số lượng sách có trong kho).
- Thực thể **Publisher** - nhà xuất bản có các thuộc tính: Publisher_ID, Publisher_name (Tên nhà xuất bản).
- Thực thể **Genre** - thể loại sách có các thuộc tính: Genre_ID, Genre_name (Tên loại sách, ví dụ: Khoa học, Ngoại ngữ,...).
- Thực thể **Author** - tác giả có các thuộc tính Author_ID và Author_name (tên tác giả).
- Thực thể **Order** có các thuộc tính Order_ID, Address (Địa chỉ giao hàng), Create_date (Ngày mua), Status (Trạng thái đơn hàng), Total_price (Tổng giá trị đơn hàng, Note (Lưu ý riêng của khách hàng).
- Thực thể **Order_detail** có các thuộc tính Price (Giá gốc), Quantity (Số lượng), Total_cost (Tổng giá = Price x Quantity), Detail_ID.
- Thực thể **Shipping_method** có các thuộc tính Shipping_ID, Fee (Giá mỗi km), Shipping_name (tên phương thức giao hàng).
- Thực thể **Payment_method** có các thuộc tính Payment_ID, Payment_name (Tên phương thức thanh toán).
- Thực thể **Cart** có các thuộc tính Cart_ID, Account_ID.
- Thực thể **Cart_detail** có các thuộc tính Cart_detail_ID, Cart_ID, Book_ID, Price (Giá cho mỗi cuốn sách), Quantity (Số lượng), Total_cost (Tổng giá).
- Thực thể **Account** có các thuộc tính Account_ID, FName (Tên), LName (Họ), Email, TelephoneNum (Số điện thoại), Password (Mật khẩu), Birthday (Sinh nhật), Address (Địa chỉ), Delete (Xóa mềm).
- Thực thể **Staff** thừa kế các thuộc tính của thực thể **Account** và có thêm các thuộc tính riêng Salary (Lương hàng tháng).
- Thực thể **Role** có các thuộc tính Role_ID, Role_name (Tên chức vụ/nhiệm vụ của nhân viên).
- Thực thể **Customer** thừa kế các thuộc tính của thực thể **Account** và có thêm các thuộc tính riêng Bank_ID (Tài khoản ngân hàng) và Bank_name (Tên ngân hàng cung cấp dịch vụ Online Banking).
- Thực thể **Reviews** là thực thể yếu, có các thuộc tính: Create_date (Ngày khách comment và đánh giá sản phẩm, Content (Nội dung đánh giá), Ratings (Bình chọn của khách hàng tối đa 5 sao), Image (Ảnh minh họa cho sản phẩm được đánh giá do khách hàng đăng tải).

4.2.4 Các ràng buộc

4.2.4.1 Ràng buộc về khóa

- Thực thể **Book** có thuộc tính khóa Book_ID.
- Thực thể **Publisher** có thuộc tính khóa Publisher_ID và thuộc tính duy nhất (Unique) Publisher_name.
- Thực thể **Genre** có thuộc tính khóa Genre_ID và thuộc tính duy nhất (Unique) Genre_name.
- Thực thể **Author** có thuộc tính khóa Author_ID và thuộc tính duy nhất (Unique) Author_name.



- Thực thể **Account** có thuộc tính khóa Account_ID.
- Thực thể **Role** có thuộc tính khóa Role_ID và thuộc tính duy nhất (Unique) Role_name.
- Thực thể **Customer** có thuộc tính duy nhất (Unique) là Bank_ID.
- Thực thể **Order** có thuộc tính khóa là Order_ID.
- Thực thể **Order_detail** có thuộc tính khóa là Detail_ID
- Thực thể **Cart** có thuộc tính khóa là Cart_ID và Account_ID là Unique.
- Thực thể **Cart_detail** có thuộc tính khóa là Cart_detail_ID.
- Thực thể **Shipping_method** có thuộc tính khóa là Shipping_ID và Shipping_name là Unique.
- Thực thể **Payment_method** có thuộc tính khóa là Payment_ID và Payment_name là Unique.
- Thực thể **Reviews** phụ thuộc vào thực thể mạnh **Account** có khóa riêng phần là Create_date.

4.2.4.2 Ràng buộc về cấu trúc

- **Cardinality ratio**

- Một tác giả (Author) có thể viết nhiều quyển sách (Book).
- Một quyển sách (Book) có thể được viết bởi nhiều tác giả (Author).
- Một quyển sách (Book) thuộc một hoặc nhiều thể loại (Genre).
- Một thể loại (Genre) có thể có nhiều quyển sách.
- Một quyển sách (Book) chỉ được xuất bản bởi một nhà xuất bản (Publisher).
- Một nhà xuất bản (Publisher) có thể xuất bản nhiều quyển sách (Book).
- Một nhân viên (Staff) phải có một chức vụ/nhiệm vụ (Role).
- Một nhiệm vụ (Role) có thể có nhiều nhân viên (Staff).
- Một tài khoản (Account) có thể viết nhiều bình luận (Reviews).
- Một Reviews được viết bởi một tài khoản (Account).
- Một quyển sách (Book) có thể có nhiều bình luận (Reviews).
- Một khách hàng (Customer) có thể đặt nhiều đơn hàng (Order).
- Một đơn hàng (Order) chỉ duy nhất thuộc về một khách hàng (Customer).
- Một đơn hàng (Order) có nhiều chi tiết đơn hàng (Order_detail).
- Một chi tiết đơn hàng (Order_detail) thuộc một đơn hàng (Order).
- Một quyển sách (Book) có thể thuộc nhiều Order_detail.
- Một Order_detail chỉ có một quyển sách (Book).
- Một đơn hàng (Order) có duy nhất một Shipping_method.
- Một Shipping_method có thể thuộc nhiều Order.
- Một đơn hàng (Order) có duy nhất một payment_method.
- Một Payment_method có thể thuộc nhiều đơn hàng (Order).
- Một Khách hàng (Customer) chỉ có một giỏ hàng (Cart).
- Một giỏ hàng thuộc về một khách hàng.
- Một giỏ hàng có thể có nhiều Cart_detail.
- Một mục trong giỏ hàng (Cart_detail) thuộc về một giỏ hàng.

- **Participation constraint**



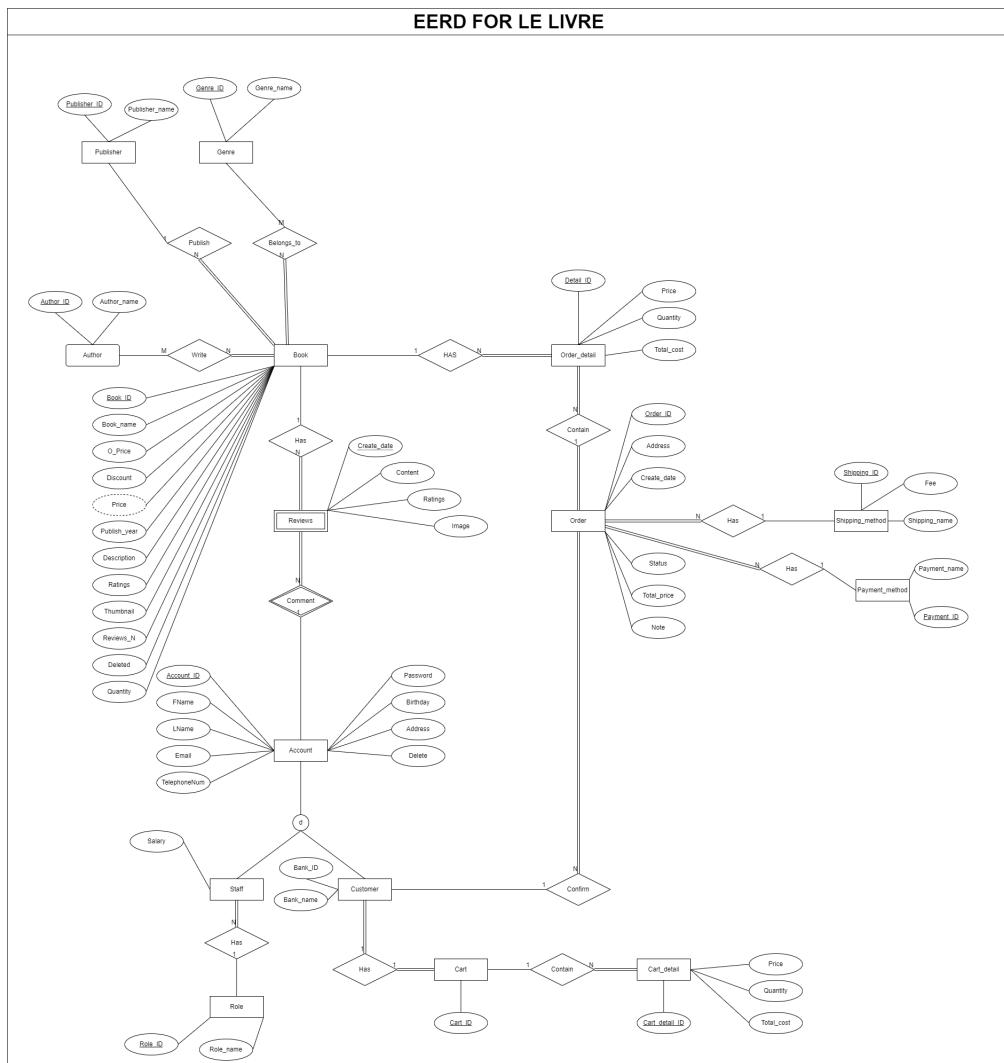
– **Mandatory (Total Participation)**

- * Một quyển sách (Book) phải được viết bởi một hoặc nhiều tác giả (Author).
- * Một quyển sách (Book) phải thuộc về một hoặc nhiều thể loại (Genre).
- * Một quyển sách (Book) phải được xuất bản bởi một nhà xuất bản (Publisher).
- * Một bình luận (Reviews) phải thuộc về một quyển sách (Book).
- * Một bình luận (Reviews) phải được viết bởi một khách hàng (Customer).
- * Một nhân viên (Staff) phải thuộc về một nhiệm vụ (Role).
- * Một Order_detail phải thuộc về một quyển sách (Book).
- * Một Order_detail phải thuộc về một Order.
- * Một Order phải chứa một hoặc nhiều Order_detail.
- * Một Order phải có một Shipping_method và một Payment_method.
- * Một khách hàng (Customer) có một giỏ hàng.
- * Một giỏ hàng (Cart) thuộc về một khách hàng.
- * Một mục giỏ hàng (Cart_detail) thuộc về một giỏ hàng.

– **Optional (Partial Participation)**

- * Một tác giả (Autho) có thể không viết quyển sách nào (Book).
- * Một thể loại (Genre) có thể không có quyển sách nào (Book).
- * Một quyển sách (Book) có thể không thuộc chi tiết đơn hàng nào (Order_detail).
- * Một quyển sách (Book) có thể không có bình luận nào (Reviews).
- * Một tài khoản (Account) có thể không viết bình luận (Reviews).
- * Một khách hàng (Customer) có thể không đặt đơn hàng nào (Order).
- * Một phương thức thanh toán (Shipping_method) có thể không thuộc đơn hàng nào (Order).
- * Một nhiệm vụ (Role) có thể không được đảm nhiệm bởi nhân viên nào (Staff).
- * Một giỏ hàng (Cart) có thể không có mục giỏ hàng nào (Cart_detail).

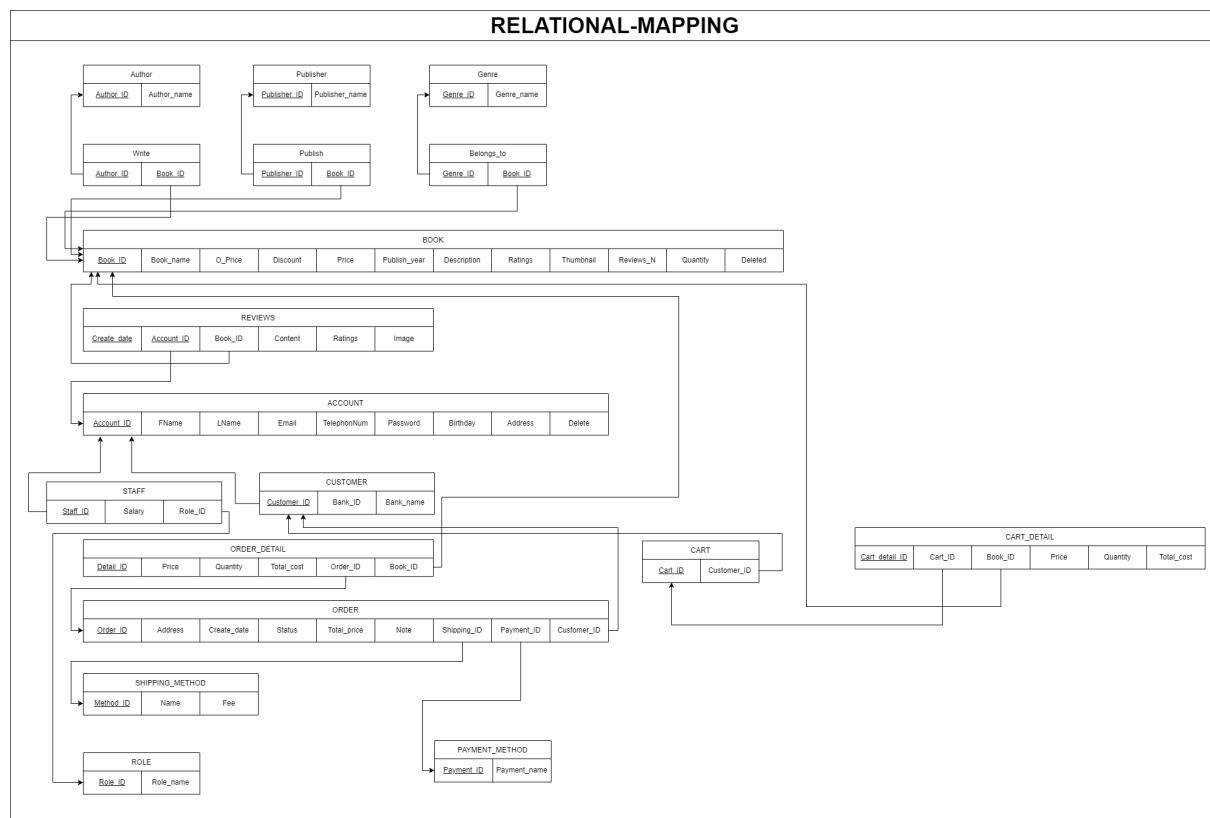
4.2.5 Thiết kế cơ sở dữ liệu ý niệm



Hình 4.1: Lược đồ E-ERD

4.2.6 Thiết kế cơ sở dữ liệu luận lý và định nghĩa các ràng buộc

4.2.6.1 Lược đồ cơ sở dữ liệu luận lý



Hình 4.2: Lược đồ cơ sở dữ liệu luận lý

4.2.6.2 Định nghĩa các ràng buộc

1. Các ràng buộc về khóa (Key constraints), toàn vẹn thực thể (Entity integrity constraints) và tham chiếu (Referential integrity constraints)

- TABLE: BOOK

- Thuộc tính Book_ID là khóa chính, không được phép NULL và có tính duy nhất.

- TABLE: AUTHOR

- Thuộc tính Author_ID là khóa chính, không được phép NULL và có tính duy nhất.

- TABLE: WRITE

- Thuộc tính Author_ID và Book_ID kết hợp làm khóa chính. Author_ID là khóa ngoại tham chiếu đến Author_ID của bảng AUTHOR và Book_ID là khóa ngoại tham chiếu đến Book_ID của bảng BOOK.

- TABLE: PUBLISHER

- Thuộc tính Publisher_ID là khóa chính, không được phép NULL và có tính duy nhất.

- TABLE: PUBLISH

- Thuộc tính Publisher_ID và Book_ID kết hợp làm khóa chính. Publisher_ID tham chiếu đến Publisher_ID của bảng PUBLISHER và Book_ID tham chiếu đến Book_ID của bảng BOOK.

- TABLE: GENRE



- Thuộc tính Genre_ID là khóa chính.

- TABLE: BELONGS_TO

- Thuộc tính Genre_ID và Book_ID kết hợp làm khóa chính. Genre_ID tham chiếu đến Genre_ID của bảng GENRE và Book_ID tham chiếu đến Book_ID của bảng BOOK.

- TABLE: ACCOUNT

- Thuộc tính Account_ID là khóa chính, không NULL và có tính duy nhất.

- TABLE: STAFF

- Thuộc tính STAFF_ID là khóa chính, đồng thời nó cũng là khóa ngoại tham chiếu đến Account_ID của bảng ACCOUNT.
- Thuộc tính Role_ID là khóa ngoại tham chiếu đến Role_ID của bảng ROLE

- TABLE: ROLE

- Thuộc tính Role_ID là khóa chính.

- TABLE: CUSTOMER

- Thuộc tính Customer_ID là khóa chính và cũng là khóa ngoại tham chiếu đến Account_ID của bảng ACCOUNT.

- TABLE: REVIEWS

- Thuộc tính Create_date và Account_ID kết hợp làm khóa chính, Account_ID là khóa ngoại tham chiếu đến Account_ID của bảng ACCOUNT.

- TABLE: CART

- Thuộc tính Cart_ID là khóa chính.
- Thuộc tính Customer_ID là khóa ngoại tham chiếu đến Customer_ID của bảng CUSTOMER.

- TABLE: CART_DETAIL

- Thuộc tính Cart_detail_ID là khóa chính.
- Thuộc tính Cart_ID là khóa ngoại tham chiếu đến Cart_ID của bảng CART.
- Thuộc tính Book_ID là khóa ngoại tham chiếu đến Book_ID của bảng BOOK.

- TABLE: ORDER

- Thuộc tính Order_ID là khóa chính.
- Thuộc tính Shipping_ID là khóa ngoại tham chiếu đến Method_ID của SHIPPING_METHOD.
- Thuộc tính Payment_ID là khóa ngoại tham chiếu đến Payment_ID của PAYMENT_METHOD.

- TABLE: ORDER_DETAIL

- Thuộc tính Detail_ID là khóa chính.
- Thuộc tính Order_ID là khóa ngoại tham chiếu đến Order_ID của bảng ORDER.
- Thuộc tính Book_ID là khóa ngoại tham chiếu đến Book_ID của bảng BOOK

- TABLE: SHIPPING_METHOD

- Thuộc tính Method_ID là khóa chính.

- TABLE: PAYMENT_METHOD

- Thuộc tính Role_ID là khóa chính.

2. Các ràng buộc ngữ nghĩa (semantic constraints) và ràng buộc miền trị

- TABLE: BOOK

- Giá gốc (O_Price) phải lớn hơn 0.
- Ratings phải nhỏ hơn hoặc bằng 5.



- Số lượng sách trong cửa hàng (Quantity) phải lớn hơn hoặc bằng 0.
- Giá bán (Price) và phần trăm khuyến mãi (Discount) phải lớn hơn hoặc bằng 0.
- Năm xuất bản của quyển sách phải nhỏ hơn hoặc bằng năm hiện tại.

- **TABLE: ACCOUNT**

- Lấy số năm hiện tại trừ đi số năm trong Birthday kết quả phải lớn hơn hoặc bằng 18.
- Password là một chuỗi có ít nhất một chữ in hoa, một số và một ký tự đặc biệt (!, @, #).

- **TABLE: STAFF**

- Salary phải lớn hơn 0.

- **TABLE: REVIEWS**

- Giá trị của Ratings phải là một số nguyên từ 1 đến 5.

- **TABLE: CART_DETAIL**

- Quantity phải lớn hơn 0.

- **TABLE: ORDER**

- Create_date phải nhỏ hơn hoặc bằng ngày giờ hiện tại.
- Total_price phải lớn hơn 0.
- Status chỉ có bốn giá trị tương ứng: 0 - Dang xử lý (Processing), 1 - Dang vận chuyển (Delivering), 2 - Vận chuyển thành công (Completed), 4 - Đơn hàng bị hủy (Canceled).

- **TABLE: ORDER_DETAIL**

- Quantity phải lớn hơn 0.
- Total_cost và Price phải lớn hơn hoặc bằng 0.

4.3 Hiện thực cơ sở dữ liệu (Database) trên Microsoft SQL Server

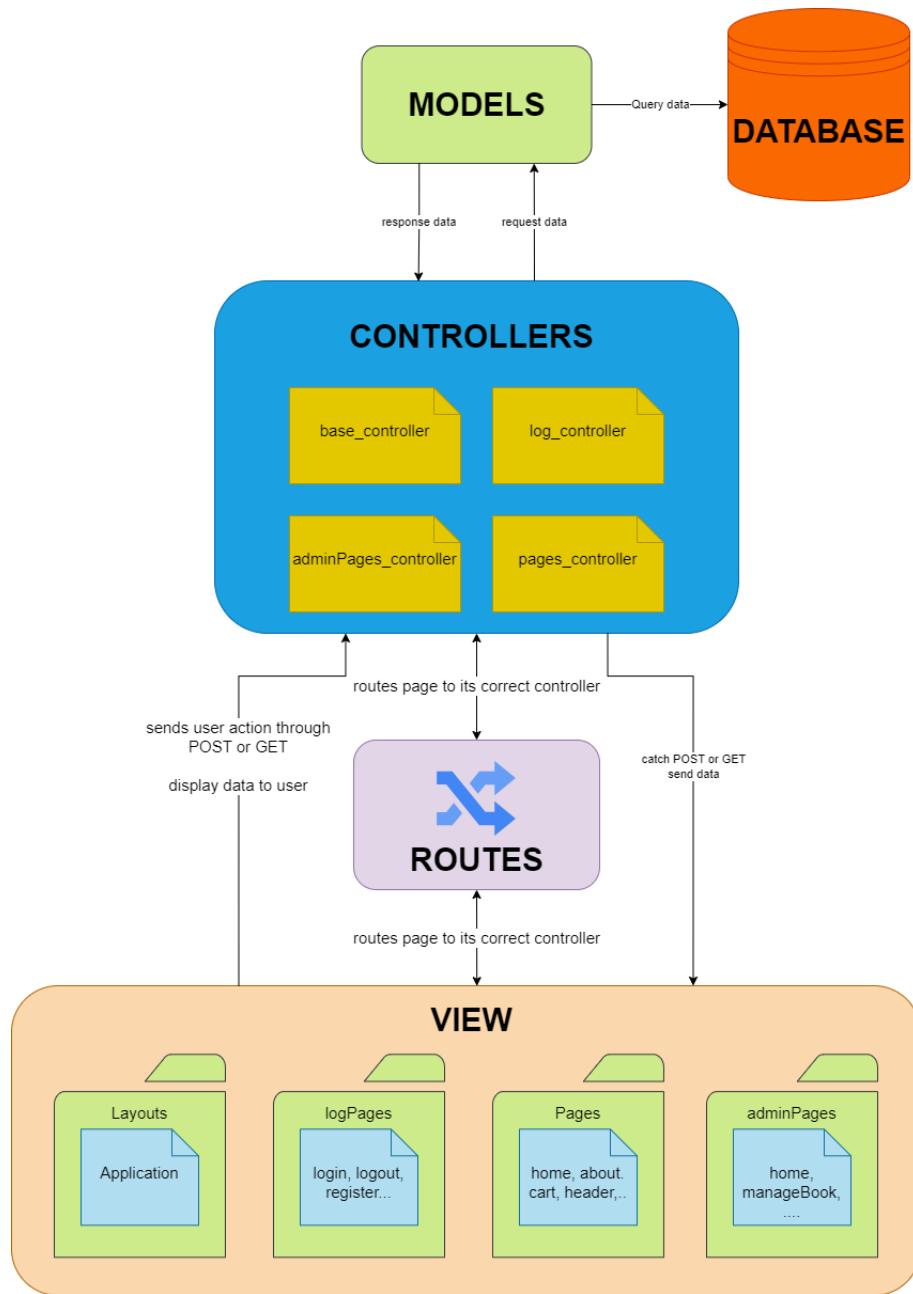
Vui lòng truy cập đường dẫn github sau để xem phần hiện thực Database link

4.4 Kiến trúc ứng dụng

4.4.1 Mô tả sơ bộ về các công nghệ sử dụng

- **Ngôn ngữ lập trình Backend, API:** Nhóm sử dụng PHP làm ngôn ngữ lập trình chính cho phần hiện thực API giao tiếp giữa giao diện và Backend. Javascript để hỗ trợ cho việc kiểm tra các ràng buộc ở tầng ứng dụng.
- **Giao diện:** Nhóm sử dụng HTML, CSS, Javascript kết hợp với framework Bootstrap V5 để hiện thực phần giao diện ứng dụng.
- **Host:** Nhóm sử dụng trình mô phỏng Xampp để host ứng dụng tại local.
- **RDBMS:** Nhóm sử dụng Microsoft SQL Server và SQLSRV Driver API để kết nối backend với cơ sở dữ liệu.

4.4.2 Mô tả sơ bộ về kiến trúc ứng dụng



Hình 4.3: Lược đồ kiến trúc ứng dụng: MVC

• Tóm tắt

Nhóm hiện thực ứng dụng dựa trên kiến trúc MVC. Các thành phần của kiến trúc được diễn đạt trừu tượng như trên. Tất cả các đường dẫn đến các trang trong ứng dụng đều có dạng `index.php?controller=p1&action=p2`, trong đó `p1` và `p2` đại diện lần lượt cho tên của controller và action(function) trong controller hay tên trang (trong ứng dụng này).

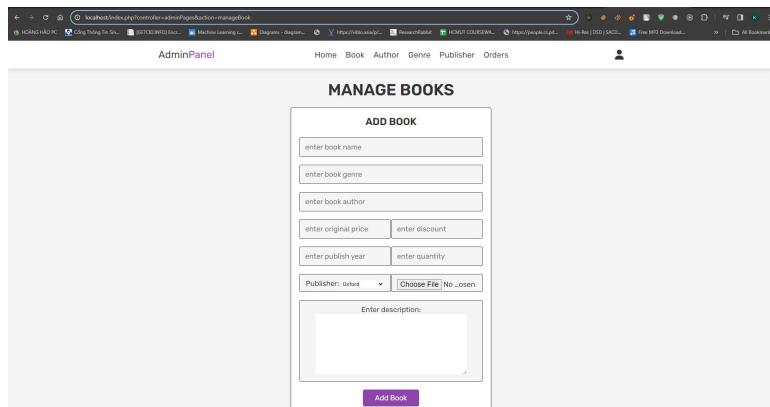
- **VIEWS**: Giao diện người dùng.
- **ROUTES**: Bộ phận điều hướng trang đến đúng với controller của nó.
- **CONTROLLERS**: Chứa các controller của ứng dụng. Ở đây được chia thành 4:

- * **base_controller:** Controller gốc, là cha của các controller khác. Chứa hàm xuất ra (render) giao diện của ứng dụng, nhận đầu vào là một dãy (array) các dữ liệu (data) cần xuất ra màn hình.
 - * **log_controller:** Dảm nhận vai trò xử lí các truy cập đến Đăng nhập (login), Đăng xuất (logout), Đăng ký (Register) của người dùng.
 - * **adminPages_controller:** Dảm nhận vai trò xử lí các truy cập từ trang Admin.
 - * **pages_controller:** Dảm nhận vai trò xử lí các truy cập từ trang User (người dùng).
- **MODELS:** Chứa các hàm truy vấn trực tiếp đến cơ sở dữ liệu.

• Demo một số trang tiêu biểu

– Trang manage books:

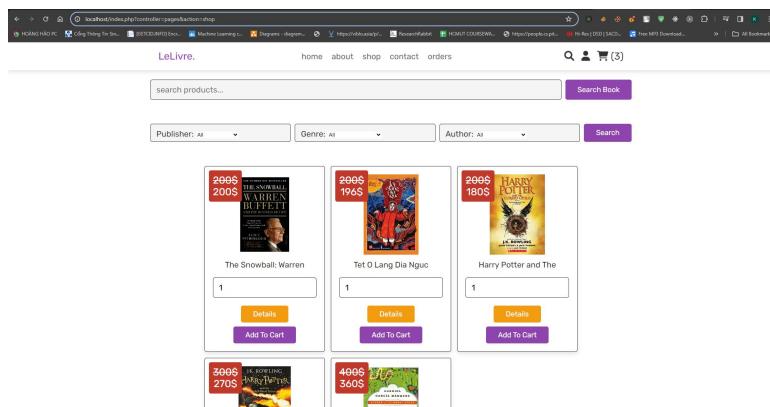
- * **Tính năng:** ADD BOOK (Thêm sách).
- * **Đường dẫn:** index.php?controller=adminPages&action=manageBook, controller là adminPages_controller và function render ra trang trong controller tên là manageBook, file giao diện của trang này tên là managebook.php.



Hình 4.4: Trang manage books

– Trang hiển thị thông tin chi tiết của sách.

- * **Tính năng:** Tìm sách, Lọc (Filter), xem danh sách các sản phẩm hiện có.
- * **Đường dẫn:** index.php?controller=pages&action=shop, controller là pages_controller và function render ra trang trong controller tên là shop, file giao diện của trang này tên là shop.php.



Hình 4.5: Trang shop



4.5 Github repository

Vui lòng truy cập đường dẫn sau để đến Github repository: link

4.6 Thiết kế và hiện thực Database trên RavenDB

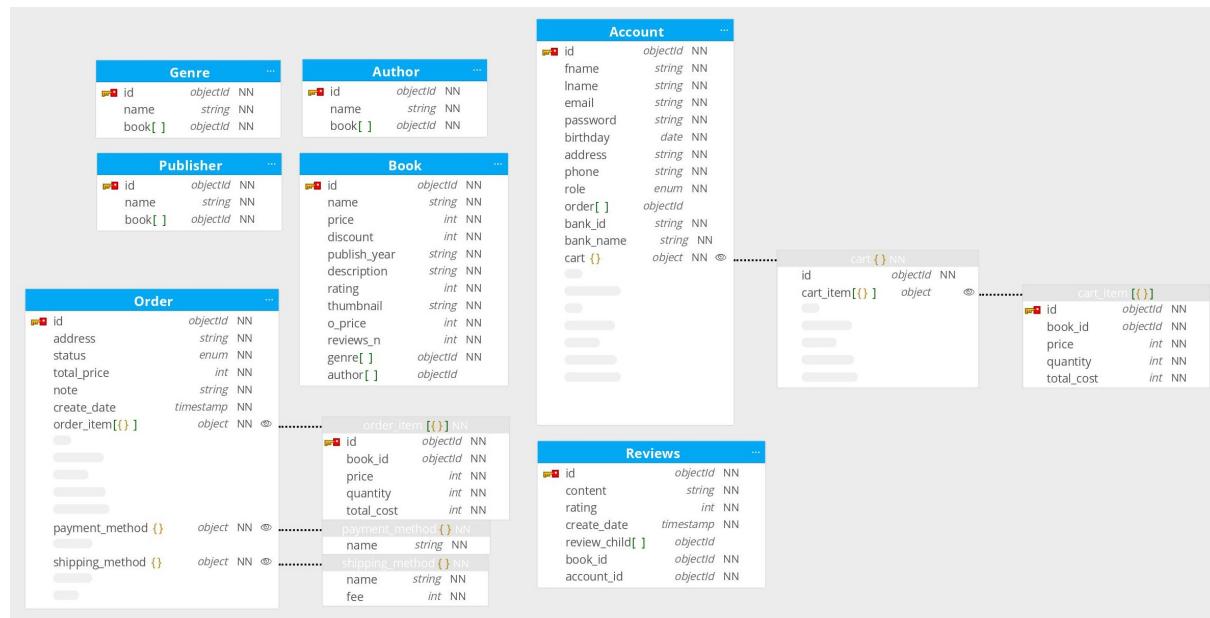
4.6.1 Mô hình hóa dữ liệu

RavenDB là hệ quản trị cơ sở dữ liệu dạng NoSQL, do đó việc thiết kế cơ sở dữ liệu không tuân theo quy tắc cụ thể hay bị ràng buộc bởi các tham chiếu như hệ quản trị cơ sở dữ liệu quan hệ. Các đối tượng dữ liệu có thể có bất kỳ thuộc tính nào, dù cho thuộc cùng 1 collection. Chính vì vậy, ta cũng không có schema cụ thể, cố định cho toàn bộ dữ liệu. Tuy nhiên, dựa vào sự phân tích và thiết kế cơ sở dữ liệu cho MS SQL trong phần trên, ta sẽ thiết kế cơ sở dữ liệu tương tự trong RavenDB. Để việc thiết kế hiệu quả hơn, một số tiêu chí sau có thể cần thiết:

Những mối quan hệ được thiết kế trong RDBMS sẽ có 2 cách tiếp cận trong NoSQL nói chung và RavenDB nói riêng:

- Embedded collection (Nhúng collection): Giữ các thực thể được tham chiếu dưới dạng collection được nhúng trên collection chính.
 - Các mục từ bộ sưu tập được nhúng không được giới thiệu ở bất kỳ đâu độc lập với bộ sưu tập chính.
 - Bộ sưu tập nhúng không được truy vấn độc lập với bộ sưu tập chính.
 - Bộ sưu tập nhúng bị giới hạn về số lượng (bounded).
- Reference collection (Tham chiếu collection): Collection chính với ID của 1 đối tượng được lưu trữ trong một đối tượng của collection riêng biệt.
 - Collection được tham chiếu ở nhiều nơi.
 - Collection được tham chiếu thường xuyên được cập nhật.

Dưới đây là lược đồ mô phỏng thiết kế cho cơ sở dữ liệu trong RavenDB:



Hình 4.6: Lược đồ mô phỏng cơ sở dữ liệu

Dữ liệu được lưu trữ trong cơ sở dữ liệu "**DBMS_RavenDB**" dưới dạng các document, với các collection tương ứng gồm **Author**, **Book**, **Genre**, **Order**, **Publisher**, **Reviews**, **Account**. Có thể thấy số lượng collection ít hơn so với số lượng table trong RDBMS. Ngoài ra, cấu trúc của mỗi collection được thiết kế sao cho đơn giản hóa việc quản lý và truy vấn dữ liệu.

Hình 4.7: Tổ chức và quản lý các document trong cơ sở dữ liệu DBMS_RavenDB

4.6.2 Hiện thực truy vấn

Để chuẩn bị cho các tính năng cho ứng dụng, dưới đây là các ví dụ cho một số lệnh truy vấn, cập nhật mà nhóm xây dựng.

RavenDB không hỗ trợ việc tạo câu lệnh insert, delete, update trực tiếp từ Studio. Do đó, ta sẽ sử dụng Client API để tương tác giữa ứng dụng client và RavenDB Cluster, từ đó tạo các lệnh này từ ứng dụng client.

Tất cả các câu lệnh truy vấn trong RavenDB đều yêu cầu có chỉ mục để hỗ trợ truy vấn tốt hơn. Vì RavenDB hỗ trợ rất tốt đối với lập chỉ mục tự động nên hầu như các truy vấn được hiện thực đều có chỉ mục tự động, mặc định sẽ sử dụng trình phân tích Exact. Các ví dụ dưới đây sẽ làm rõ việc đánh chỉ mục tự động bởi trình tối ưu hóa sẽ được áp dụng cho truy vấn như thế nào.

4.6.2.1 Lệnh Insert

Trước khi thực hiện các câu lệnh insert, delete, update, cần tạo Document Store để quản lý tương tác giữa ứng dụng client và RavenDB Cluster. Tiếp đó tạo lập 1 Session từ Document Store giúp biểu diễn các operation này trên database.

```
1  using (var store = new DocumentStore())
2  {
3      /* Define the Session's options object */
4      SessionOptions options = new SessionOptions()
5      {
6          Database = "DBMS_RavenDB",
7          TransactionMode = TransactionMode.ClusterWide
8      };
9
10     /* Open the Session in a Synchronous mode */
11     /* Pass the options object to the session */
12     using (IDocumentSession session = store.OpenSession(options))
13     {
14         session.SaveChanges();
15     }
16 }
```

Thêm 1 quyển sách, trong đó id() được tạo tự động.

```
1  -- generate Id automatically
2  session.Store(new Book
```



```
3     name = 'The Alchemist',
4     o_price = 800000,
5     discount = 50,
6     publish_year = 2000,
7     description = 'Helps you dare to live your dreams, without fear of failure',
8     rating = 5,
9     thumbnail = 'new_alchemist.jpg',
10    reviews_N = 50,
11    quantity = 100;
12
13 /* send all pending operations to server, in this case only 'Put' operation */
14 session.SaveChanges();
```

Entity document mới được thêm vào collection book.

4.6.2.2 Lệnh Delete

Xóa 1 quyển sách với id cho trước.

```
1 -- load entity book from collection
2 Book book = session.Load<Book>("book/1");
3 session.Delete(book);
4 -- save changes to session
5 session.SaveChanges();
```

4.6.2.3 Lệnh Update

Trong RavenDB, id() là thuộc tính quan trọng, được thiết lập khi tạo document và không thể thay đổi được nên ta chỉ có thể chỉnh sửa các thuộc tính khác của document.

Chỉnh sửa thông tin 1 quyển sách với id cho trước.

```
1 /* Load a entity document */
2 /* The entity loaded from the document will be added to the Session's entities map*/
3 Book book = session.Load<Book>("book/1");
4 /* Update the book's information */
5 book.name = 'The Hobbit';
6 book.o_price = 700000;
7 book.discount = 80;
8 book.publish_year = '2016';
9 book.description = 'A fantasy novel by J.R.R. Tolkien';
10 book.rating = 5;
11 book.thumbnail = 'hobbit.jpg';
12 book.reviews_n = 550;
13 book.quantity = 70;
14 /* Apply changes */
15 session.SaveChanges();
```

4.6.2.4 Truy vấn với single condition

Trong hệ thống bán sách, việc truy xuất sách theo từng nhóm phân loại giúp người dùng dễ dàng tìm kiếm và nhận biết rõ hơn về phân loại sách trong hệ thống.

Lệnh truy vấn sách thuộc thể loại 'Fiction', RavenDB sử dụng chỉ mục tự động Map khi ta không chỉ định chỉ mục cho truy vấn và không có chỉ mục phù hợp. Đối với field có liên quan giữa 2 collection genre

và book là Id() được lưu trong key book của các document trong collection Book, RavenDB cho phép người dùng sử dụng lệnh load() để load nội dung của document có Id tương ứng ra key-value trong kết quả

```
1 from genre as g
2 where name = 'Fiction'
3 select {
4     books: load(g.book)
5 }
```

The screenshot shows the RavenDB Query Studio interface. In the top-left, there's a 'Query' section containing the following code:

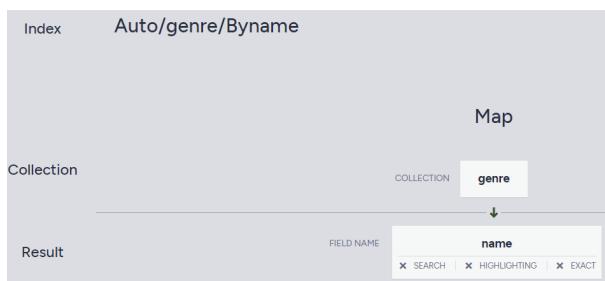
```
1 from genre as g
2 where name = 'Fiction'
3 select {
4     books: load(g.book)
5 }
```

In the top-right, there are buttons for 'Save...', 'Load Query', 'Settings', and a play button. Below the query editor, the 'Results' section displays the following message: 'Index 'Auto/genre/Byname' was used to get the results in 44 ms'. It shows a table with two columns: 'Preview' and 'Id'. Under 'Preview', there's a small icon. Under 'Id', it shows 'genre/2'. On the right side of the results table, there are buttons for 'Delete documents', 'Copy to clipboard', and a dropdown menu.

A portion of the JSON result is visible on the right, showing a single document with fields like 'name', 'price', 'discount', 'publish_year', 'description', 'rating', 'thumbnail', 'category', and 'genre'.

Hình 4.8: Kết quả truy vấn

Chỉ mục tự động Map 'Auto/genre/ByName' trên trường name được xác định cho truy vấn này. RavenDB tìm kiếm dựa trên các Term của chỉ mục, sử dụng tùy chọn tìm kiếm Exact và trả về kết quả là 1 document trong 44ms.



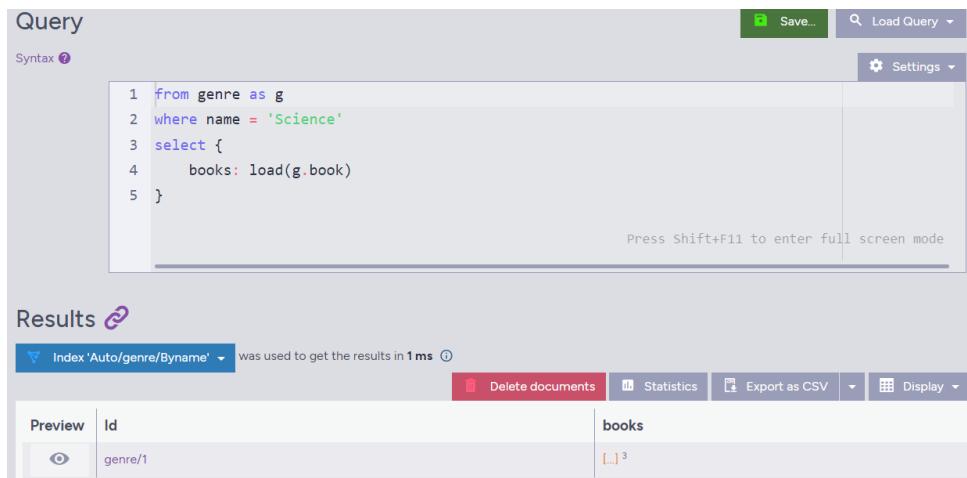
Hình 4.9: Mô phỏng chỉ mục

The screenshot shows the 'Index terms for Auto/genre/ByName' interface. It lists two sections: 'name' and 'id'. The 'name' section shows two terms: 'fiction' and 'science', each with a '2 loaded' status. The 'id' section shows two terms: 'genre/1' and 'genre/2', also with a '2 loaded' status.

Hình 4.10: Index Term

Vì đây là lần đầu tiên thực hiện truy vấn mà không chỉ định chỉ mục cụ thể nên thời gian có kết quả chậm. Thời gian truy vấn bao gồm thời gian tìm kiếm chỉ mục phù hợp, tạo chỉ mục tự động cho truy vấn và thời gian chạy chỉ mục cho ra kết quả. Tuy nhiên, sau truy vấn này, thực hiện truy vấn tương tự

với phân loại 'Science', RavenDB sẽ nhận dạng chỉ mục đã có 'Auto/genre/ByName' là chỉ mục phù hợp cho truy vấn. Do đó, thời gian truy vấn giảm đi đáng kể, chỉ còn 1ms.



The screenshot shows the RavenDB Studio interface. In the 'Query' tab, a simple LINQ-style query is written:

```

1 from genre as g
2 where name = 'Science'
3 select {
4     books: load(g.book)
5 }

```

In the 'Results' tab, the output is displayed. It shows a single document with the ID 'genre/1'. The 'books' field contains three items, indicated by a small orange square icon with the number '3'.

Hình 4.11: Kết quả truy vấn tương tự

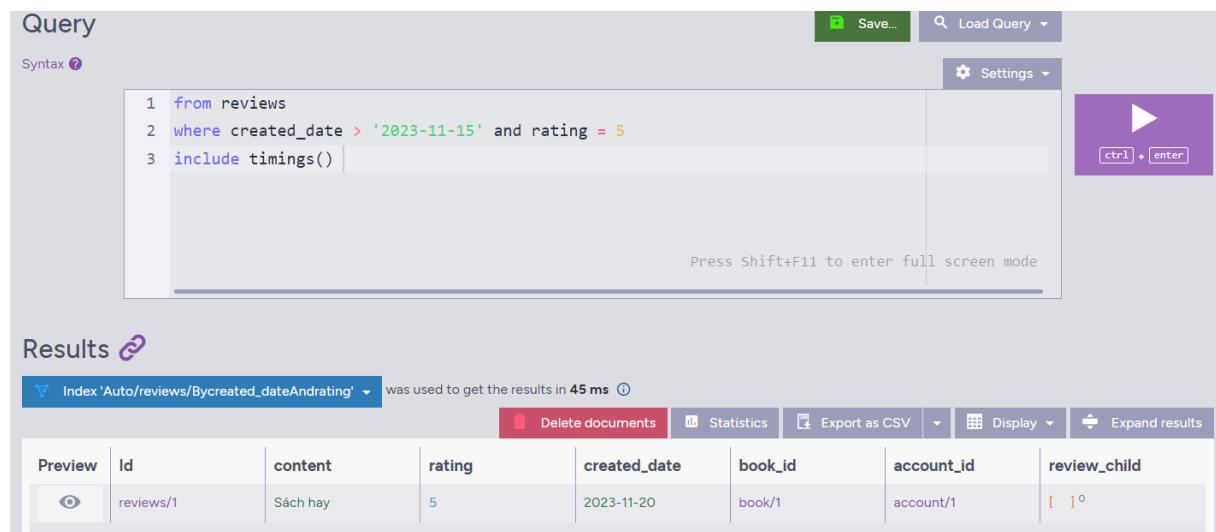
4.6.2.5 Truy vấn với composite condition

Truy vấn thông tin đánh giá(REVIEWS) được tạo(Create_date) sau ngày 2023-11-15 và có đánh giá(Ratings) 5 sao:

```

1 from reviews
2 where created_date > '2023-11-15' and rating = 5

```



The screenshot shows the RavenDB Studio interface. In the 'Query' tab, a query is written using the 'include' keyword to include timing information:

```

1 from reviews
2 where created_date > '2023-11-15' and rating = 5
3 include timings()

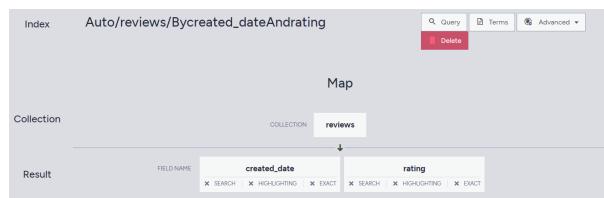
```

A purple button on the right says 'ctrl + enter' with a play icon, indicating the query can be run.

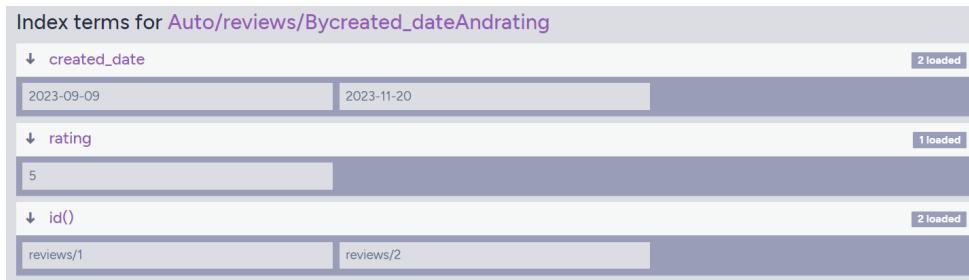
In the 'Results' tab, the output is displayed. It shows a single document with the ID 'reviews/1'. The document contains fields: content ('Sách hay'), rating (5), created_date ('2023-11-20'), book_id ('book/1'), account_id ('account/1'), and review_child ([] 0).

Hình 4.12: Kết quả truy vấn

Chỉ mục tự động Map 'Auto/reviews/Bycreated_dateAndrating' trên trường created_date và rating được xác định cho truy vấn này. RavenDB tìm kiếm dựa trên các Term của chỉ mục, sử dụng tùy chọn tìm kiếm Exact và trả về kết quả là 1 document trong 45ms.



Hình 4.13: Mô phỏng chỉ mục



Hình 4.14: Index Term

4.6.2.6 Truy vấn với join

Khi hiển thị chi tiết 1 sản phẩm sách, tên tác giả và thể loại là điều tất yếu để cung cấp thông tin cụ thể của sách.

Theo logic của SQL, điều trên có nghĩa là thực hiện truy vấn thông tin sách trên collection book, join với collection author và collection genre. Tuy nhiên RavenDB không join các collection dựa trên field có liên quan giữa chúng. Đối với field có liên quan là Id() của 1 collection, RavenDB cho phép người dùng sử dụng lệnh load() để load nội dung của document có Id tương ứng ra 1 cặp key-value trong kết quả.

Lệnh truy vấn thông tin cụ thể của sách có Id là 'book/1'.

```
1 from "book" as b
2 where id() = 'book/1'
3 load b.genre as g[], b.author as a[]
4 select [
5   Name: b.name,
6   Year: b.publish_year,
7   Genres: g.map(i => i.name),
8   Authors: a.map(i => i.name)
9 ]
```



Document: book/1 X

```
{  
    "Name": "DBMS",  
    "Year": "2020",  
    "Genres": [  
        "Science",  
        "Fiction"  
    ],  
    "Authors": [  
        "Xuân Hồng",  
        "Nguyễn Nhật Ánh"  
    ],  
    "@metadata": {  
        "@id": "book/1",  
        "@last-modified": "2023-11-26T06:09:28.6966647Z",  
        "@change-vector": "A:11-04u1kiyJT0KMRMvHUPHRGg",  
        "@projection": true,  
        "@index-score": 0  
    }  
}
```

Close Copy to Clipboard

Hình 4.15: Kết quả truy vấn

Thời gian truy vấn là 0ms cho thấy việc truy vấn với RavenDB trên trường Id rất nhanh chóng.

4.6.2.7 Truy vấn với subquery

Truy vấn yêu cầu tìm các cuốn sách vừa có giá thành lớn hơn 100000 và đánh giá lớn hơn mức 2.

```
1 from "book" as b  
2 where intersect(b.price > 100000, b.rating > 2)
```

Index Auto/book/BypriceAndpublish_yearAndrating` was used to get the results in 0 ms ⓘ										
Preview	Id	name	price	discount	publish_year	description	rating	thumbnail	o_price	review_n
	book/3	Testing	500,000	10	2015	Testing book	3		400,000	0
	book/2	Xin chào ngày mới	250,000	5	2004	Sách cũ	4		200,000	0
	book/4	Coding	300,000	10	2015	Coding book	3		400,000	0

Hình 4.16: Kết quả truy vấn

4.6.2.8 Truy vấn với aggregate function

Truy vấn yêu cầu tìm tổng giá trị các cuốn sách gộp theo năm phát hành.

```
1 from "book" as b  
2 group by b.publish_year  
3 select [  
4     "Year": b.publish_year
```



```
5     "Total": sum(b => b.price)
6 }
```

Index 'Auto/book/BypriceReducedBypublish_year'		was used to get the results in 2 ms ⓘ	Delete documents	Statistics	Export as CSV	Display	Expand results
Preview	Year	Total					
	2015	820,000					
	2020	100,000					
	2004	250,000					

Hình 4.17: Kết quả truy vấn

CHƯƠNG 5

TỔNG KẾT

5.1 Đường dẫn video báo cáo và repository ứng dụng

- Video báo cáo
- Source code ứng dụng

5.2 Tóm tắt công việc đã làm

Trong nội dung báo cáo, nhóm đã thực hiện tìm hiểu tổng quan và trình bày 5 chức năng của hai DBMS Microsoft SQL Server và RavenDB:

- Indexing
- Query processing và optimization
- Transaction processing
- Concurrency control
- Backup và recovery

Về việc hiện thực truy vấn, với mỗi DBMS, nhóm đã trình bày 8 loại truy vấn khác nhau từ đơn giản đến phức tạp. Nhóm cũng đã hiện thực thành công một ứng dụng chạy trên DBMS Microsoft SQL Server.

5.3 Đánh giá kết quả đạt được

5.3.1 Đối với Microsoft SQL Server

Qua tìm hiểu, bản thân nhóm đánh giá Microsoft SQL Server là một hệ quản trị cơ sở dữ liệu được thiết kế rất chi tiết và tỉ mỉ phục vụ cho nhiều nghiệp vụ khác nhau, rất đáng để phân tích và bàn luận xa hơn. Trong khuôn khổ và ở ngoài phạm vi Bài Tập Lớn này, vẫn còn rất nhiều tính năng độc đáo của Microsoft SQL Server ta chưa tìm hiểu hết. Bản thân nhóm ngoài việc sử dụng Documentation đến từ chính đội ngũ Microsoft, cũng đã tìm hiểu và tham khảo thêm nhiều tài liệu chuyên sâu, ít phổ biến hơn trong cộng đồng nhằm mang đến những thông tin và góc nhìn mới mẻ nhất cho Bài Tập Lớn này. Mong giảng viên, người đọc có thể thấy được điều này, thông cảm cũng như góp ý về những thiếu sót hay sự không nhất quán trong một số nội dung.



5.3.2 Đối với RavenDB

Bên cạnh SQL Server, RavenDB cũng là 1 hệ quản trị cơ sở dữ liệu mạnh mẽ, cho phép chúng ta thực hiện gần như đầy đủ các tính năng như 1 RDBMS. Nó hỗ trợ không chỉ về chỉ mục mà còn về trình truy vấn, giao tác và cả điều khiển tương tranh. Điều này cho thấy các nhà phát triển rất cố gắng trong việc tạo ra 1 công cụ quản trị hiệu quả nhất cho người dùng. Điểm nổi bật của RavenDB có thể kể đến như:

- Truy vấn nhanh chóng nhờ việc lập chỉ mục cho tất cả các truy vấn, đặc biệt chỉ mục tự động hỗ trợ rất tốt cho người dùng.
- Sử dụng trình tối ưu hóa truy vấn phân tích (query optimizer) với vai trò là **xác định những chỉ mục** nào sẽ được sử dụng cho truy vấn cụ thể này.
- Sử dụng mô hình Change vector để kiểm soát những ván đè liên quan đến Concurrency
- Sử dụng đồng thời bản sao lưu và snapshot, kết hợp với bản sao lưu gia tăng (an incremental backup) để hỗ trợ cho việc giảm gánh nặng lưu trữ các bản sao lưu và recovery được nhanh chóng

Đó là những đánh giá của nhóm sau khi tìm hiểu về RavenDB. Vì thời gian có hạn nên có thể còn nhiều thiếu sót trong các nội dung đã trình bày ở trên. Những tìm hiểu trong học kỳ qua sẽ là nền tảng giúp nhóm có thêm kiến thức không chỉ về RavenDB mà còn hiểu thêm về các hệ quản trị NoSQL.

TÀI LIỆU THAM KHẢO

- [1] dbForge Team.
SQL Server Architecture: An In-Depth Guide.
<https://blog.devart.com/sql-server-architecture.html#protocol>.
- [2] Oren Eini.
Inside RavenDB Book.
<https://ravendb.net/learn/inside-ravendb-book/reader/4.0>, 2018.
- [3] Esat Erkec.
Query optimization techniques in SQL Server: tips and tricks.
<https://www.sqlshack.com/query-optimization-techniques-in-sql-server-tips-and-tricks/>.
- [4] Esat Erkec.
Transactions in SQL Server for beginners.
<https://www.sqlshack.com/transactions-in-sql-server-for-beginners/>.
- [5] Steve Hall.
Explicit Transactions.
<https://www.sqlservercentral.com/articles/explicit-transactions>.
- [6] Microsoft.
SQL Server technical documentation.
<https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16>.
- [7] RavenDB.
NoSQL Database Documentation.
<https://ravendb.net/docs/article-page/6.0/csharp>.
- [8] Ian Sommerville.
Software Engineering (10th ed.).
Pearson, 2016.
ISBN: 978-0133943030.
- [9] Itamar Syn-Hershko.
RavenDB in Action.
<https://livebook.manning.com/book/ravendb-in-action-free/foreword/>, 2017.