**PHYSYCAL DATABASE DESIGN AND TUNING**

## 16.7.3 Tinh chỉnh truy vấn và khung nhìn:

Khi tinh chỉnh một câu truy vấn, điều đầu tiên cần xác định là hệ thống đang sử dụng lược đồ chúng ta muốn sử dụng hay không. Có thể hệ thống không thể tìm thấy cách tiến hành tốt nhất vì một vài lí do nào đó. Một vài trường hợp thông thường không được xử lý hiệu quả bởi các bộ tối ưu là:

* Điều kiện chọn liên quan đến giá trị null
* Điều kiện chọn liên quan đến biểu thức số học hoặc chuỗi hoặc điều kiện sử dụng liên kết OR. Ví dụ, nếu có một điều kiện E.age = 2 \* D.age trong mệnh đề WHERE, bộ tối ưu có thể sử dụng đúng index có trên E.age nhưng index trên D.age thì không. Nếu sử dụng E.age /2 = D.age thì kết quả ngược lại
* Không thể nhận biết một sơ đồ tinh vi như phép quét chỉ trên index đối với một truy vấn tập hợp sử dụng GROUP BY. Tất nhiên, không có bộ tối ưu nào sẽ tìm kiếm một sơ đồ bên ngoài không gian sơ đồ được mô tả trong chương 12, 13 ví dụ như cây phép kết không có nhánh trái (nonleft-deep join trees). Vì thế, thấu hiểu được những gì một bộ tối ưu điển hình làm là một điều quan trọng. Thêm vào đó, bạn càng nhận thức được sức mạnh và hạn chế của một hệ thống, sẽ càng tốt cho bạn

Nếu một bộ tối ưu không đủ thông minh để tìm được cách thực hiện tốt nhất, một vài hệ thống cho phép người dùng hướng dẫn chọn lựa một sơ đồ bằng cách cung cấp gợi ý cho bộ tối ưu, ví dụ người dùng có thể can thiệp vào việc sử dụng một index cụ thể hoặc chọn thứ tự phép kết và phương thức kết. Nhưng để làm được như vậy, người dùng phải thấu hiểu cả bộ tối ưu và khả năng của DBMS

# 16.8 Sự lựa chọn trong tinh chỉnh lược đồ quan niệm:

Chúng ta sẽ sử dụng ví dụ sau để minh họa cho các chọn lựa trong tinh chỉnh lược đồ quan niệm:

*Contracts (cid: integer, supplierid: integer, projectid: integer, deptid: integer, partid: integer, qty: integer, value: real)*

*Departments(did: integer, budget: real, annualreport: varchar)*

*Parts(pid: integer, cost: integer)*

*Projects(jid: integer, mgr:char(20))*

*Suppliers (sid: integer, address: char(50))*

Để ngắn gọn, chúng ta sẽ dùng những qui ước thông thường của thuộc tính kí hiệu bằng một kí tự đơn, và lược đồ quan hệ kí hiệu bằng dãy các kí tự. Xem xét lược đồ quan hệ Contracts, chúng ta sẽ kí hiệu là CSJDPQV, với mỗi kí tự ký hiệu cho một thuộc tính. Ở đây có 2 ràng buộc toàn vẹn liên quan đến Contracts: JP 🡪 C, và SD 🡪 P

## 16.8.1 Sự bố trí dạng chuẩn thấp hơn:

Xem xét quan hệ Contracts. Chúng ta có nên phân rã nó ra những quan hệ nhỏ hơn không ? Ở đây quan hệ Contracts đạt dạng chuẩn 3.

Bằng cách sử dụng PTH SD 🡪 P ta có thể phân rã thành 2 quan hệ con là SDP và CSJDQV. Cách phân rã này bảo toàn thông tin nhưng không bảo toàn phụ thuộc hàm. Tuy nhiên, bằng ách sử dụng lược đồ quan hệ CJP, chúng ta đạt được một phân rã không mất kết hợp và bảo toàn phụ thuộc hàm. Đây là phương án tốt, chúng ta có thể thay Contracts bằng 3 quan hệ con CJP, SDP, CSJDQV

Tuy nhiên, giả sử câu truy vấn sau thường được thực hiện: Tìm số bản sao Q của P được sắp xếp trong hợp đồng C. Câu truy vấn đòi hỏi một phép kết của quan hệ CJP và CSJDQV (hoặc SDP và CSJDQV), trong khi đó nó có thể trả lời trực tiếp nếu sử dụng quan hệ Contracts. Chi phí cộng thêm của câu truy vấn này có thể thuyết phục chúng ta sử dụng một thiết kế dạng chuẩn 3NF và không phân rã Contracts.

## 16.8.2 Denormalization:

Lí do thúc đẩy chúng ta thiết lập một dạng chuẩn thấp có thể dẫn chúng ta đến những bước sâu hơn: đưa vào dư thừa có tính toán. Như trong ví dụ, giả sử một câu truy vấn thường xuyên kiểm tra giá trị của một hợp đồng có ít hơn ngân sách hợp đồng của phòng ban hay không. Chúng ta có thể quyết định thêm một cột ngân sách B vào Contracts. Bởi vì did là khóa của Departments, chúng ta sẽ có PTH D 🡪 B trong Contracts, có nghĩa là Contracts không ở dạng chuẩn 3 nữa. Dù sao, chúng ta có thể chọn cách thiết kế này vì nếu thúc đẩy truy vấn là quan trọng. Một quyết định như vậy rõ ràng là chủ quan và dẫn đến chi phí dư thừa đáng kể.

## 16.8.3 Sự chọn lựa phân rã:

Xem xét quan hệ Contracts lần nữa, có vài chọn lựa có thể giải quyết vấn đề dư thừa dữ liệu trong quan hệ này:

* Chúng ta có thể giữ quan hệ Contracts như vậy và chấp nhận dư thừa dữ liệu của nó ở dạng chuẩn 3.
* Chúng ta có thể phân rã Contracts tới dạng chuẩn BCNF nếu muốn tránh các bất thường trong dư thừa dữ liệu:
  + Sử dụng cách phân rã trên bảo toàn thông tin và bảo toàn PTH bằng cách phân rã thành 3 quan hệ con SDP, CSJDQV, CJP
  + Chỉ phân rã ra 2 quan hệ SDP, CSJDQV dù nó không bảo toàn PTH

Cách phân rã 2 quan hệ SDP, CSJDQV không ngăn cản chúng ta tuân theo PTH JP 🡪 C, nó chỉ khiến điều này trở nên đắt giá hơn. Chún g ta có thể sử dụng assertion để kiểm tra ràng buộc này

*CREATE ASSERTION checkDep*

*CHECK (NOT EXISTS (SELECT \**

*FROM PartInfo PI, ContractInfo CI*

*WHERE PI.supplierid = CI.supplierid AND PI.deptid = CI.deptid*

*GROUP BY CI.projectid, PI.partid*

*HAVING COUNT(cid) >1 ))*

Assertion này tốn chi phí đánh giá hơn vì nó liên quan đến phép kết theo sau bởi thao tác sắp xếp. Đối với cách còn lại, hệ thống có thể kiểm ra JP là khóa chính của bảng CJP bằng index trên JP. Điểm khác biệt này trong chi phí kiểm tra ràng buộc là thúc đẩy giữ lại phụ thuộc hàm. Mặt khác, nếu cập nhật không thường xuyên, chi phí tăng thêm có thể chấp nhận được, vì thế chúng ta có thể chọn không duy trì bảng CJP

## 16.8.4 Phân rã chiều dọc:

Giả sử chúng ta đã quyết định phân rã Contracts thành SDP, CSJDQV. Những lược đồ này đều đạt BCNF và không ó lí do gì để phân rã tiếp. Tuy nhiên, giả sử những truy vấn sau được sử dụng thường xuyên:

* Tìm những hợp đồng được kí bởi nhà cung cấp S
* Tìm những hợp đồng được đặt ở phòng ban D

Các truy vấn trên sẽ khiến ta phân rã CSJDQV thành CS, CD vad CJQV. Cách phân rã này tất nhiên không mất mát, và 2 câu truy vấn trên có thể được trả lời bằng cách kiểm tra ở những quan hệ nhỏ hơn

Bất cứ khi nào phân rã một quan hệ, chúng ta cần xem xét những câu truy vấn nào mà phân rã đó sẽ ảnh hưởng bất lợi, đặc biệt khi nếu lí do duy nhất để phân ra là để cải thiện hiệu suất. Ví dụ nếu truy vấn tổng giá trị các hợp đồng được kí bởi nhà cung cấp, nó sẽ liên quan đến phép kết các quan hệ đã phân rã CS, và CJQV. Trong trường hợp này, chúng ta lại quyết định ngược lại với cách phân rã.

## 16.8.5 Phân rã chiều ngang:

Chúng ta đã tìm hiểu về phân rã theo chiều dọc ở trên, đôi khi nên xem xét thay thế một quan hệ bằng 2 quan hệ có cùng thuộc tính như quan hệ ban đầu, mỗi quan hệ chứa một tập con của các bộ trong quan hệ ban đầu. Phương pháp này được gọi là phân rã chiều ngang, một cách trực giác phương pháp này hiệu quả khi các tập con của các bộ khác nhau được truy vấn bằng những cách khác nhau

Ví dụ quan hệ Contracts có thể được thay thế bằng 2 quan hệ LargeContracts - SmallContracts, một quan hệ chứa các bộ có giá trị hợp đồng lớn hơn 10000 – LargeContracts, một quan hệ chứa các bộ còn lại - SmallContracts nếu các truy vấn trên hợp đồng có giá trị lớn hơn 10000 và phần còn lại khác nhau. Nếu sử dụng cách này, chúng ta có thể tạo một view là tổ hợp của LargeContracts và SmallContracts chính là Contracts ban đầu

# 16.9 Sự lựa chọn trong tinh chỉnh truy vấn và khung nhìn:

Bước đầu tiên trong tinh chỉnh truy vấn là thấu hiểu sơ đồ được sử dụng bởi DBMS để đánh giá câu truy vấn. Khi đã hiểu được, chúng ta sẽ biết cách tinh chỉnh hiệu suất như thế nào. Chúng ta có thể xem xét một sự lựa chọn khác của index hoặc có thể cluster kết hợp giữa 2 qun hệ kết với nhau trong truy vấn, được hướng theo cách hiểu của chúng ta về sơ đồ cũ, và lược đồ tốt hơn mà ta muốn DBMS sử dụng. Chi tiết tương tự quá trình thiết kế khởi tạo

Một điều đangs để làm là trước khi tạo idex mới chúng ta nên xem xét viết lại truy vấn có thể đạt kết quả chấp nhận được với index hiện tại hay không. Ví dụ xem xét truy vấn sau với toán tử OR:

*SELECT E.dno*

*FROM Employees E*

*WHERE E.hobby = ‘Sampts’ OR E.age = 10*

Nếu ta đặt index trên cả hobby và age, chúng ta có thể sử dụng index để thu thập các bộ cần thiết, nhưng bộ tối ưu hóa có thể không nhận biết được trong trường hợp này. Bộ tối ưu hóa sẽ xem xét điều kiện trong mệnh đề WHERE trên tổng thể không có index nào, thực hiện quét tuần tự trên Employeé, và áp dụng chọn lựa khi đang hoạt động. Giả sử khi chúng ta viết lại câu truy vấn trên sử dụng phép hội giữa 2 câu truy vấn, một với mệnh đề WHERE E.hobby = ‘Stamps’ và một với WHERE E.age = 10. Khi đó mỗi truy vấn sẽ đáp ứng tốt hơn với sự giúp đỡ của index trên hobby và age.

Chúng ta nên cân nhắc viết lại truy vấn tránh các phép toán tốn nhiều chi phí. Ví dụ DISTINCT trong mệnh đề SELECT sẽ tốn nhiều chi phí. Vì vậy nên bỏ DISTINCT bất cứ khi nào có thể. Ví dụ, đối với truy vấn trên một quan hệ đơn lẻ, chúng ta có thể bỏ DISTINCT bất cứ khi nào những điều kiện sau xảy ra:

* Chúng ta không quan tâm đến sự có mặt của những bộ trùng lắp
* Thuộc tính đề cập trong mệnh đề SELECT bao gồm khóa ứng viên của quan hệ

Đôi khi một truy vấn có GROUP BY và HAVING có thể được thay thế bởi truy vấn không có những mệnh đề đó, do đó loại bỏ được phép toán sắp xếp. Ví dụ:

*SELECT MIN(E.age)*

*FROM Employees E*

*GROUP BY E.dno*

*HAVING E.dno = 102*

Câu truy vấn tương đương

*SELECT MIN(E.age)*

*FROM Employees E*

*WHERE E.dno = 102*

Truy vấn phức tạp thường được viết theo từng bước, sử dụng các quan hệ tạm thời. Chúng ta có thể viết lại mà không có quan hệ tạm thời để chúng thực hiện nhanh hơn. Ví dụ:

*SELECT \**

*INTO Temp*

*FROM Employees E, Department D*

*WHERE E.dno = D.dno AND D.mgrname = ‘Robinson’*

*SELECT T.dno, AVG(T.sal)*

*FROM Temp T*

*GROUP BY T.dno*

Câu truy vấn có thể viết lại như sau:

*SELECT E.dno, AVG(E.sal)*

*FROM Employees E, Department D*

*WHERE E.dno = D.dno AND D.mgrname = ‘Robinson’*

*GROUP BY E.dno*

Câu truy vấn trên không cụ thể hóa quan hệ trung gian Temp nên thực hiện nhanh hơn. Sự thật là bộ tối ưu thậm chí có thể tìm thấy sơ đồ hiệu quả chỉ có index mà không cần thu thập các bộ Employees nếu có cây index B+ dày đặt và phức hợp trên (dno, sal). Ví dụ này minh họa một quan sát tổng quát: bằng cách viết lại truy vấn tránh những tạm thời không cần thiết, chúng ta không chỉ tránh tạo các quan hệ tạm thời, chúng ta còn mở ra nhiều khả năng tối ưu hơn cho bộ tối ưu thực hiện

Trong một vài trường hợp, tuy nhiên, nếu bộ tối ưu không thể tìm ra một lược đồ tốt cho truy vấn phức tạp (điển hình là một truy vấn lồng tương quan), nên viết lại truy vấn sử dụng quan hệ tạm thời để hướng bộ tối ưu đến một lược đồ tốt.

Sự thật là, truy vấn lồng là một tài nguyên thông thường của tính không hiệu quả, bởi vì nhiều bộ tối giải quyết nghèo nàn với chúng. Bất cứ khi nào có thể, tốt hơn là viết lại truy vấn lồng không sử dụng lồng, và viết truy vấn tương quan mà không có tính tương quan. Như đã lưu ý, một sửa đổi tốt của truy vấn có thể đòi hỏi chúng ta đưa vào những quan hệ mới tạm thời và kĩ thuật thực hiện như vậy một cách có hệ thống (một cách lý tưởng là được thực hiện bởi bộ tối ưu) đã được nghiên cứu rộng rãi.

# 16.10 Tác động của tính đồng thời:

Trong hệ thống với nhiều người dùng đồng thời, một vài điều mở rộng nên được xét đến. Mỗi chương trình người dùng (giao tác) sẽ thực hiện các lock trên các trang nó đọc/viết. Những giao tác khác không thể truy xuất trên trang bị lock đến khi giao tác đó hoàn thành và giải phóng lock. Hạn chế này có thể dẫn đến sự tranh chấp lock gay gắt trên các trang đã dùng

Khoảng thời gian mà giao tác giữ lock có thể ảnh hưởng đáng kể đến hiệu suất. Tinh chỉnh giao tác bằng cách viết các biến chương trình cục bộ và trì hoãn các thay đổi đến cơ sở dữ liệu đến khi kết thúc giao tác có thể cải thiện đáng kể hiệu suất. Hiệu suât có thể được cải thiện bằng cách thay thể giao tác bằng một vài giao tác nhỏ, mỗi giao tác có thể giữ lock trong khoảng thời gian ngắn

Ở mức độ vật lý, phân chia cẩn thận các bộ trong quan hệ và index liên quan trên các đĩa có thể cải thiện đáng kể hiệu năng truy xuất đồng thời. Ví dụ, nếu ta có một quan hệ trên một đĩa và một index trên đĩa khác, những truy xuất đến index có thể được tiếp tục mà không cần can thiệp những truy xuất đến quan hệ, ít nhất là ở mức độ đọc đĩa.

Nếu một quan hệ được cập nhật thường xuyên, cây index B+ cụ thể có thể trở thành một hiện tượng thắt cổ chai khi truy cập đồng thời bởi vì nhiều truy xuất thông qua index phải đi qua gốc, vì vậy gốc và trang index bên dưới nó trở thành điểm nóng, nghĩa là những trang đó chịu tranh chấp gay gắt. Nếu hệ quản trị cơ sở dữ liệu sử dụng những giao thức lock chuyên biệt đối với cây index, và cụ thể là sử dụng lock ở mức chi tiết (fine-granularity), vấn đề này sẽ được giảm bớt. Nhiều hệ thống luồng sử dụng nhiều kỹ thuật. Tuy nhiên, những cân nhắc này khiến ta chọn ISAM index trong một vài trường hợp. Bởi vì mức index của một ISAM index là tĩnh, chúng ta không cần lock trên những trang này, chỉ có trang lá cần được lock. Một ISAM index có thể thích hợp hơn với cây B+, ví dụ, nếu cập nhật thườn xuyên xảy ra nhưng ta mong muốn sự phân phối cân đối các record và số lượng của record với miền cho trước của các giá trị khóa tìm kiếm ở mức xấp xỉ nhau. Trong trường hợp này, ISAM index đưa ra chi phí lock thấp hơn (giảm tranh chấp lock), và sự phân phối các record sao cho ít trang bị quá tải hơn sẽ được thực hiện

Cách cập nhật một quan hệ có thể trở nên quan trọng. Ví dụ, nếu một bộ được chèn vào quan hệ Emplyees theo thứ tự eid và chúng ta có index B+ trên eid, mỗi giá trị chèn vào sẽ nằm ở trang lá cuối cùng của cây B+. Điều này dẫn đến điểm nóng dọc con đường từ gốc đến trang lá bên trái cùng. Các suy xét khiến ta chọn một hash index trên cây index B+ hoặc index trên một field khác. (Chú ý cách truy xuất này cũng có thể dẫn đến hiệu năng kém cho index ISAM, bởi vì trang lá cuối cùng trở thành điểm nóng)

Ngoài ra đây không phải là vấn đề đối với hash index bởi vì xử lý băm chọn ngẫu nhiên vùng chứa record được chèn vào

Các đặc điểm của SQL cho các tính chất của giao tác xác định có thể được dùng để cải thiện hiệu năng. Nếu một giao tác không điều chỉnh cơ sở dữ liệu, chúng ta có thể xác định access mode là READ ONLY. Đôi khi có thể chấp nhận cho một giao tác có thể nhìn thấy một vài dữ liệu bất thường do thực thi đồng thời. Đối với những giao tác đó, nhiều tính đồng thời hơn có thể đạt được bằng cách điều khiển tham số được gọi là isolation level.

# 16.11 Đánh giá hệ quản trị cơ sở dữ liệu:

Khi đánh giá một sản phẩm DBMS, hiệu suất là suy xét quan trọng nhất. Môtk DBMS là một phần phức tạp của phần mềm, và những nhà cung cấp khác nhau có thể hướng hệ thống của họ đến những thị phần khác nhau bằng cách đặt nặng tối ưu hóa các phần nhất định của hệ thống, hoặc bằng cách chọn lọc nhữn thiết kế hệ thống khác nhau. Ví dụ, một vài hệ thống được thiết kế để chạy các câu truy vấn phức tạp một cách hiệu quả, trong khi đó các hệ thống khác lại thực thi nhiều giao tác đơn giản trong một giây. Trong mỗi loại hệ thống, có nhiều sản phẩm cạnh tranh. Để hỗ trợ người dùng chọn lựa DBMS thích hợp với nhu cầu của họ, các công cụ đánh giá hiệu suất được phát triển. Các tiêu chuẩn đo lường hiệu suất của một lớp các ứng dụng (như tiêu chuẩn TPC) và tiêu chuẩn đo lường mức độ thực thi đa dạng các phép toán của một DBMS (ví dụ tiêu chuẩn Winconsin)

Các tiêu chuẩn nên linh động, dễ hiểu và đánh giá một cách tự nhiên đối với các biểu hiện lỗi. Chúng nên đo lường hiệu suất đỉnh (như số giao tác / giây, hoặc gọi là tps) cũng như các tỉ lệ chi phí / hiệu năng (như $/tps) đối với lượng công việc điển hình trong phạm vi ứng dụng cho trước. Hội đồng xử lý giao tác được thành lập để định nghĩa các tiêu chuẩn cho xử lý giao tác và hệ thống cơ sở dữ liệu. Những tiêu chuẩn nổi tiếng khác được đề nghị bởi các nhà nghiên cứu hàn lâm và các tổ chức công nghiệp. Tiêu chuẩn là tài sản đối với nhà cung cấp và thường không hữu dụng khi so sánh các hệ thống khác nhau (mặc dù chúng có ích khi xác định khả năng xử lý lượng công việc cụ thể của một hệ thống cho trước)

## 16.11.1 Các tiêu chuẩn DBMS nổi tiếng:

***Tiêu chuẩn xử lý giao tác online (TPC):*** tiêu chuẩn TPC-A và TPC-B thiết lập định nghĩa chuẩn của đơn vị tps và $/tps. TPC-A đo lường hiệu năng và chi phí của một mạng máy tính bổ sung với DBMS trong khi TPC – B chỉ xem xét DBMS. Những tiêu chuẩn gồm một giao tác đơn giản cập nhật 3 record dữ liệu, từ 3 bảng khác nhau, và chèn thêm 1 record vào bảng thứ 4. Số lượng chi tiết (ví dụ phân phối giao tác đến, phương thức liên kết, tính chất hệ thống) được xác định một cách khắt khe, đảm bảo kết quả cho mỗi hệ thống có thể được so sánh có ý nghiax. Tiêu chuẩn TPC-C là một bộ phức tạp các hoạt động giao tác hơn TPC – A, TPC –B. Nó mô hình hóa một kho hàng lưu các mặt hàng cung cấp cho khách hàng và chứa 5 loại giao tác. Mỗi giao tác TPC-C đắt hơn nhiều so với giao tác TPC – A và TPC – B, TPC – C kiểm tra một phạm vi rộng hơn trên khả năng của hệ thống như sử dụng index thứ cấp và hủy bỏ giao tác. Nó có nhiều hơn hoặc ít thay thế hoàn toàn TPC – A và TPC – B như một tiêu chuẩn giao tác xử lý cơ bản

***Tiêu chuẩn truy vấn:*** Tiêu chuẩn Wisconsin được sử dụng rộng rãi để đo lường hiệu suất của các truy vấn quan hệ cơ bản. Tiêu chuẩn Set Query đo lường hiệu suất của một bộ các truy vấn phức tạp hơn, và tiêu chuẩn đo hiệu suất kết hợp các tải (workload) của giao tác, truy vấn quan hệ, và các hàm hữu ích. Tiêu chuẩn TPC-D là một bộ các truy vấn SQL phức tạp, được đại diện cho phạm vi ứng dụng hỗ trợ ra quyết định. Hội đồng OLAP còn phát triển một tiêu chuẩn cho truy vấn phức tạp hỗ trợ ra quyết định, bao gồm một số các truy vấn không thể biểu diễn trong SQL một cách dễ dàng, có ý định để đo hệ thống xử lý phân tích online (OLAP), hơn là hệ thống SQL truyền thống. Tiêu chuẩn Sequoia 2000 được thiết kế để so sánh hỗ trợ DBMS cho hệ thống thông tin địa lý

***Tiêu chuẩn cơ sở dữ liệu đối tượng:*** Tiêu chuẩn 001 và 007 đo hiệu năng của hệ thống cơ sở dữ liệu hướng đối tượng. Tiêu chuẩn Bucky đo hiệu năng của hệ thống cơ sở dữ liệu quan hệ đối tượng.

## 16.11.2 Sử dụng tiêu chuẩn:

Tiêu chuẩn nên được sử dụng với sự thấu hiểu chúng được thiết kế để đo lường cái gì và môi trường ứng dụng nào mà một DBMS được sử dụng. Khi sử dụng các tiêu chuẩn để chọn lựa một hệ quản trị cơ sở dữ liệu, hãy nhớ các hướng dẫn sau:

***How meaningful is a given benchmark?(*** một tiêu chuẩn cho trước có ý nghĩa như thế nào?) Tiêu chuẩn cố gắng chia nhỏ hiệu suất thành những con số đơn lẻ có thể đơn giản quá mức. Một hệ quản trị cơ sở dữ liệu là một phần phức tạp của phần mềm được sử dụng trong đa dạng các phần mềm. Một tiêu chuẩn tốt nên có một bộ các thao tác được lựa chọn kĩ lưỡng để bao phủ phạm vi ứng dụng cụ thể và để kiểm tra các đặc điểm quan trọng của hệ quản trị cơ sở dữ liệu đối với phạm vi đó

***How well does a benchmark reflect your workload?*** (Một tiêu chuẩn phản lượng công việc của bạn tốt như thế nào?) Bạn nên xem xét lượng công việc mong muốn và so sánh chúng với tiêu chuẩn. Đưa ra nhiều trọng lượng đối với hiệu suất của công việc đánh giá (ví dụ truy vấn, cập nhật) tương tự với những công việc quan trọng trong lượng công việc của bạn. Cũng xem các số tiêu chuẩn được đo như thế nào. Ví dụ, thời gian tiêu tốn của một truy vấn độc lập có thể bị lừa nếu xem xét trong một thiết lập đa người dùng: một hệ thống có thể có nhiều thời gian tiêu tốn hơn vì I/O chậm hơn. Trên một workload đa người dùng, các đĩa I/O song song cho trước, như một hệ thống có thể hoạt động tốt hơn một hệ thống với thời gian tiêu tốn thấp hơn

***Create your own bechmark*** (Tạo ra chính tiêu chuẩn của bạn): Các nhà cung cấp thường tinh chỉnh hệ thống của họ bằng cách đặc biệt để đạt được những số liệu tốt trên các tiêu chuẩn quan trọng. Để chống lại điều này, hãy tạo ra tiêu chuẩn của chính bạn bằng cách chỉnh sửa một ít tiêu chuẩn cơ bản hoặc thay thế các công việc trong một tiêu chuẩn cơ bản với các công việc tương tự từ lượng công việc của bạn