|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Họ tên** | **MSSV** | **Hình ảnh** | **Ghi chú** |
| 1 | Hà Thị Phương Thảo | 0912430 | Description: 0912430 |  |
| 2 | Nguyễn Thị Thanh Thảo | 0912431 | Description: 0912431 |  |
| 3 | Trương Nguyễn Thủy Tiên | 0912463 | Description: 0912463 |  |
| 4 | Nguyễn Văn Tiến | 0912469 | Description: 0912469 | Nhóm trưởng |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Nội dung công việc** | **Thành viên 1**  **0912430** | **Thành viên 2**  **0912431** | **Thành viên 3**  **0912463** | **Thành viên 4**  **0912469** |
| 1 | Dịch tài liệu |  |  |  |  |
| 2 | Tổng hợp kết quả, viết tài liệu |  |  |  |  |

**PHYSICAL DATABASE DESIGN AND TUNING**

1. **Dẫn nhập**

Câu lệnh trong DBMS như các truy vấn ( queries) thường gặp và các thao tác cập nhật(update) điển hình là thước đo cuối cùng về chất lượng của một thiết kế cơ sở dữ liệu. 1 DBA có thể cải tiến câu lệnh bằng cách điều chỉnh 1 vài thông số trong DBMS(ví dụ: kích thược của vùng nhớ đệm hoặc tầng số của thực thi của checkpoint) hay là tìm ra những giới hạn về khả năng đáp ứng của phẩn cứng(bottlenecks) sau đó sẽ tìm giải pháp thêm những phần cứng để đáp ứng cho vận hành hệ thống. Tuy nhiên, để đạt được thực thi câu lệnh tốt thì thiết kế dữ liệu trước tiên cần phải tốt.

sau khi thiết kế lược đồ ở mức quan niệm(conceptual and external) là tạo ra những quan hệ và những khung nhìn cùng với những ràng buộc toàn vẹn, ta phải xác định mục đích câu lệnh trong thiết kế mức vật lý(thiết kế lược đồ vật lý). Khi yêu cầu người dùng cao, thiết kế lược đồ mức vật lý rất cần thiết để hoà hợp, điều chỉnh, tất cả các khía cạnh của 1 thiết kế cơ sở dữ liệu cho câu lệnh thực thi tốt.

* ***Mục đích của chương này****:*
* *phần II.1 cho cách nhìn tổng quát về thiết kế và tinh chỉnh ở mức vậy lý. Giải quyết vấn đề liên quan đến chỉ mục (index) là điều rất quan trọng trong thiết kế mức vật lý.*
* *phần II.2 đưa ra những nguyên tắc cho việc quyết định lựa chọn tạo chỉ mục.*
* *Phần II.3 đưa ra những ví dụ làm nổi bật những cách lựa chọn index cơ bản.*
* *Phần II.4 đặt vấn đề sâu hơn với những vấn đề gom nhóm (clustering): làm thế nào để chọn index gom nhóm và có lưu các bộ từ các quan hệ khác nhau gần với chúng không.*
* *Phần II.5 bàn về việc dùng index với multiple-attribute search keys .*
* *Phần II.6 ta nhấn mạnh làm cách nào lựa chọn tốt index có thể cho phép nhiều câu queries được thực hiện mà không cần xem những dữ liệu có thật( actual data records).*
* *Phần II.7 nói sơ qua về tinh chỉnh dữ liệu và các bước cần làm trong tinh chỉnh.*
* *phần II.8 cách tinh lọc lại lược đồ mức quan niệm trong.*
* *phần II.9 cách tinh chỉnh lại câu queries và khung nhìn.*
* *Phần II.10**nói ngắn gọn sự ảnh hưởng lớn về các câu lệnh thực thi xảy ra đồng thời.*
* *Phần II.11 kết thúc chương với bàn luận ngắn về những tiêu chuẩn(benchmarks) trong DBMS; tiêu chuẩn lựa chọn DBMS để giúp thực thi tốt các câu lệnh.*
* ***Công cụ thiết kế vật lý****: cho đến nay RDBMSs (relational Database Management System) đãcung cấp vài công cụ để hỗ trợ việc thiết kế và tinh chỉnh lược đồ mức vật lý. MS SQL server có công cụ tinh chỉnh mà hỗ trợ về tạo index; nó cũng hỗ trợ xoá bỏ 1 index. IBM DB2 V6 cũng có công cụ tinh chỉnh và Oracle Expert hỗ trợ về thêm, xoá index.*

1. **Nội dung**
2. **giới thiệu về thiết kế mức vật lý.**

Giống như các mức thiết kế dữ liệu khác, thiết kế vật lý phải phụ thuộc phù hợp với những dữ liệu có ngoài đời thực và những dụng ý sử dụng nó. Một cách riêng, nó rất quan trọng việc nhận biết những công việc điển hình cần phải làm *(typical workload)* mà cơ sở dữ liệu phải cung cấp cho người dùng. Wordload bao gồm các câu truy vấn hoặc cập nhật dữ liệu . điều người dùng luôn quan tâm là tốc độ của câu truy vấn, cập nhật phải nhanh tới mức nào, hay là có bao nhiêu transaction phải thực thi mỗi giây. Miêu tả của workload *(workload description)* và yêu cầu về thực thi của người dùng là những yêu cầu cơ bản mà mà cần phải giải quyết trong thiết kế mức vật lý.

Để tạo 1 dữ liệu mức vật lý tốt và để tinh chỉnh hệ thống cho câu lệnh trong việc đáp trả lại yêu cầu của người dùng, người thiết kế cần phải hiểu công việc của 1 DBMS, đặc biệt là về chỉ mục và cách thực thi câu query của DBMS đó. Nếu dữ liệu mong đợi được truy cập đồng thời bởi nhiểu ngưởi dùng hoặc là 1 cơ sở dữ liệu phân tán(distributed database), nhiệm vụ trở nên phức tạp rất nhiều và các tính năng khác của DBMS được dùng đến. ta bàn luận ảnh hưởng lớn về truy cập đồng thơi trên dữ liệu được thiết kế trong 16.10.

* 1. ***database workloads (khối công việc cần làm trong dữ liệu)***
* 1 workload description bao gồm các thành phần sau:
  + *1 danh sách các query và tần số sử dụng chúng, như 1 phần nhỏ của tất cả query và update.*
  + *1 danh sách các câu lệnh update và tần số sử dụng nó.*
  + *Mục đích câu lệnh cho mỗi loại query và update.*
* Mỗi câu query trong workload, ta phải xác định:
* *Những quan hệ nào được dùng đến.*
* *Những thuộc tính nào được xuất ra (trong lệnh SELECT).*
* *Những thuộc tính nào có điều kiện lựa chọn hoặc là có điều kiện kết trong lệnh WHERE và chọn lựa những điều kiện này như thế nào.*
* Mỗi lệnh update trong workload, ta phải xác định
* *Những thuộc tính nào có điều kiện lựa chọn hoặc là điều kiện kết (trong lệnh where) và chọn lựa những điều kiện này như thế nào.*
* *Loại của câu lệnh update náy (insert , delete, hay update) và quan hệ cần update.*
* *Các trường mà bị điểu chỉnh lại trong lệnh update này.*

**Chú ý**:

* *câu lệnh query và update thường có các đối số. giá trị của những đối số này xác định độ chọn lọc của điều kiện lựa chọn và điều kiện kết*.
* Lệnh cập nhật có 1 thành phần query mà được dùng đề tìm các bộ cần update. Các câu truy vấn có mặt chỉ mục sẽ tốt, nhưng lưu ý đến ảnh hưởng cửa chỉ mục đến lệnh update.
  1. ***Physical Design and Tuning Decisions (giải quyết việc thiết kế và tinh chỉnh)***

Những giải quyết quan trong suốt quá trình thiết kế và tinh chỉnh dữ liệu bao gồm các bước sau:

* Những index nào được tạo.
  + *Những quan hệ nào được chỉ mục và những thuộc tính nào hoặc tập hợp thuộc tính nào được chọn làm khoá tìm kiếm của chỉ mục.*
  + *Mỗi chỉ mục, nó nên được gom nhóm hay không gom nhóm(unclustered), nó có nên được nhiều (dense) hay thưa thớt (sparse).*
* Ta có nên thay đổi lược đồ mức quan niềm để cải thiện truy vấn không. Ví dụ, ta phải xét điều sau:
  + *Lựa chọn lược đồ đã đạt dạng chuẩn: ta luôn có nhiều cách để phân tích 1 lược đồ thành dạng chuẩn mong muốn(BCNF hoặc 3NF). Lựa chọn có thể được dựa vào tiêu chuẩn thực thi.*
  + *Sự không tiêu chuẫn hoá (denormalization): có thể ta cần xem xét lại phân rã lược đồ thành dạng chuẩn trong suốt quá trình thiết kế lược đồ mức quan niệm để cải tiến câu lệnh query mà chứa các thuộc tính từ các quan hệ bị phân rã trước đây.*
  + *Sự phân chia thẳng đứng(vertical partitioning): ở những tình huống dĩ nhiên ta có thể muốn phân rã các quan hệ thành nhiều quan hệ nhỏ hơn nữa để tăng tốc độ thực thi của query vì nó chỉ thực hiện trên ít thuộc tính.*
  + *Khung nhìn: ta có thể tạo nhiều khung nhìn để che đi những thay đổi trong lược đồ quan hệ không cho người dùng biết.*
* Có nên thường xuyên thực thi query và transaction được ghi lại nhiều lần để tôc độ nhanh hơn không.
  1. ***Sự cần thiết của tinh chỉnh dữ liệu***

Thông thường, thông tin chi tiết của workload có thể khó nhận biết được trong khi bắt đầu thiết kế của hệ thống. do đó, điều chỉnh dữ liệu sau khi thiết kế và triển khai thì rất quan trọng- ta phải tinh chỉnh lại thiết kế ban đầu trong đúng với cách diễn tả của đời thực để thu được những câu lệnh thực thi tốt nhất có thể.

Sự khác biệt giữa thiết kết dữ liệu và tinh chỉnh dự liệu là 1 vài thứ không bị bó buộc. ta có thể xét quy trình thiết kế hơn 1 lần lược đồ ban đầu ở mức khái niệm được thiết kế và tập hợp các chỉ mục và giải quyết vấn đề gom nhóm được thực hiện. bất cứ sự thay đổi xảy ra sau khi đã tạo lược đồ mức quan niệm hoặc chỉ mục rồi thì ta có thể gọi đó là việc tinh chỉnh. Như một lựa chọn, ta có thể xét 1 vài cách tinh chỉnh lược đồ mức quan niệm là 1 phần của quá trình thiết kế vật lý.

Ranh giới giữa việc thiết kế và tinh chỉnh không cần thiết phải rõ ràng ( nó không quan trọng), và ta sẽ bàn luận 1 cách đơn giản vấn để lựa chọn chỉ mục và tinh chỉnh dữ liệu mà không cần quan tâm đến khi nào việc tinh chỉnh được thực hiện.

1. **Hướng dẫn cho lựa chọn index**

Các nguyên tắc để lựa chọn chỉ mục như sau:

* **Guideline 1 (whether to index):** quan điểm rõ ràng thường là điều quan trọng nhất. không xây dựng chỉ mục trừ khi nhiều câu truy vấn - bao gồm cả những thành phần query trong lệnh update- mà rất có ích, hay dùng đến. Bất cứ khi nào có thể, chọn những chỉ mục nào mà làm tăng tốc độ cho nhiều câu truy vấn.
* **Guideline 2 ( choice of search key):** những thuộc tính được đề cập trong lệnh WHERE là những ứng cử viên lựa chọn làm chỉ mục.
  + Điều kiện lựa chọn trong truy vấn cho ta biết nên xét 1 chỉ mục trên những thuộc tính đã được chọn 1 chỉ mục loại hash index.
  + Đối với những điều kiện thuộc loại điều kiện phạm vi (range select ) thì ta nên xét đến chỉ mục B+ tree (hoặc ISAM) trên những thuộc tính được lựa chọn. 1 chỉ mục B+ tree luôn tốt hơn 1 chỉ mục ISAM.
* **Guideline 3 (multiple-attribute search keys):** những chỉ mục với khoá tìm kiếm đa thuộc tính nên được xét trong tình huống sau:
  + Lệnh WHERE chứa các điều kiện có hơn 1 thuộc tính trong quan hệ.
  + Chúng chỉ cho phép chiến lược định lượng chỉ mục ( i.e.., việc truy cập quan hệ có thể được tránh) cho những câu query quan trọng. (tình huống này có thể dẫn đến những thuộc tính đang là khoá tìm kiếm thậm chí chúng không xuất hiện trong lệnh WHERE).

Khi tạo những chỉ mục trên các khoá tìm kiếm với nhiều thuộc tính, nếu dãy query được mong đợi , hãy cẩn thận với những thuộc tính trong khoá tìm kiếm để thực hiện query.

* **Guideline 4 (whether to cluster):** ở hầu hết 1 chỉ mục trong 1 quan hệ có thể được gom nhóm, và việc gom nhóm làm ảnh hưởng lớn đến lệnh thực thi; vì thế lựa chon gom nhóm chỉ mục rất quan trọng.
* **Guideline 5 (hast versus tree index):** 1 chỉ mục B+ tree thì luôn được chuộng hơn bởi vì nó hỗ trợ cho dạng truy vấn range query(truy vấn có liên quan đến phạm vi giá trị thuộc tính) cũng như các truy vấn bằng(equality query). 1 hash *index (A  Hash index stores key value pairs based on a pseudo randomizing function called hash function)* thì tốt hơn trong trường hợp sau:
  + Chỉ mục được thiết kế để hỗ trợ chỉ mục mà được lồng vào các phép kết lặp; quan hệ có chỉ mục là quan hệ nội bộ, và khoá tìm kiếm chứa các cột mà thực hiện phép kết. trong trường hợp này, hash index sẽ làm tăng tốc độ thực thi hơn B+ tree cho câu lệnh select bằng(equality selections) , bởi vì 1 lệnh select bằng (equality selection) được sinh ra cho mỗi bộ trong quan hệ ngoài vùng.
  + Có câu lệnh bằng(equality query) rất quan trọng, và câu lệnh đó không có truy vấn dạng range queries, bao hàm cả những thuộc tính khoá tìm kiếm.
* **Guideline 6 (balancing the cost of index maintenance):** sau khi tạo được những index như mong đợi thì hãy xét các ảnh hưởng của mỗi index trong những lệnh update nằm trong workload như thế nào.
  + Nếu việc giữ lại các chỉ mục mà gây ra thực thi chậm cho các lệnh update thì nên gõ bỏ nó đi.
  + Tuy nhiên, luôn nhớ rằng việc thêm 1 chỉ mục cũng có thể làm tăng tốc các lệnh udate. Ví du : 1 chỉ mục trên thuộc tính employee ids có thể làm tăng tốc cho thao tác lệnh: tăng lương cho các employee

1. **1 số ví dụ cơ bản cho việc lựa chọn chỉ mục**

Ví dụ dưới đây minh hoạ làm thế nào lựa chọn các chỉ mục trong quá trình thiết kế dữ liệu. lược đồ được dùng trong ví dụ này thì không diễn tả chi tiết; thông thường chúng bao gồm các thuộc tính named trong lệnh query. Thông tin thêm được lấy ra khi cần thiết.

Ta hãy bắt đầu với lệnh query đơn giản sau:

SELECT E .ename, D.mgr

FROM Employees E, Departments D

WHERE D.dname = ‘Toy’ AND D.dno = D.dno

Những quan hệ được đề cập đến trong câu query này là Employees và Departments, và 2 điều kiện trong WHREW đều là loại điều kiện bằng. theo những nguyên tắc đã nói thì ta nên xây dựng những hash index trên thuộc tính dname của Departments. Nhưng xét đẳng thức E.dno = D.dno, ta có nên tạo 1 chỉ mục (dĩ nhiên là hash index) trên 2 thuộc tính của Departments hoặc Employees( hay cả 2) không? Theo trực giác thì ta nên lấy những bộ của Departments sử dụng chỉ mục trên dname vì có ít bộ thoả mãn đẳng thức D.dname = ‘Toy’. Vì cứ tướng ứng với những bộ mà thoả mãn đẳng thức trên thì ta phải dùng đến index trên thuộc tính dno trong Employees tương ứng với những bộ trong Departments đó. Vì thế, ta nên xây dựng 1 chỉ mục trên trường dno của Employees. Lưu ý là tốc độ ttăng thêm bởi việc xây dựng index trên trường dno của Departments bởi vì các bộ của Departments mà được lấy ra thì sử dụng chỉ mục dname)

Lựa chọn những chỉ mục của chúng ta được chỉ dẫn bởi 1dự định ước lượng truy vấn (query evaluation plan) mà ta cần dùng. Sự cân nhắc của ước lượng dự định tiềm năng này là chung chung trong khi quyết định thiết kế dữ liệu. việc hiểu sự tối ưu hoá câu query là rất quan trọng cho thiết kế vật lý. Ta thể hiện dự định mong muốn cho query này trung hình 16.1.

Giả sử câu query trên có 1 chút thay đổi ở WHERE như sau: D.dname = ‘Toy AND E.dno = D.dno And E.age = 25. Chúng ta hãy xét lựa chọn những dự định ước lượng. 1 dự định tốt là để lấy ra những bộ Departments mà thoả với đẳng thức dname và lấy ra những hộ Employees bằng cách dùng chỉ mục trên trường dno; việc chọn age được ứng dụng on-the-fly. Tuy nhiên, không giống như query trước, ta không cần có 1 chỉ mục trên trường dno của Employees nếu ta có 1 chỉ mục trên age. Trường hợp này ta có thể lấy lấy các bộ Departments mà thoả mãn đẳng thức trên dname(bằng cách dùng index trên dname, như trước đó). Lấy các bộ Employees mà thoả mãn các lựa chọn trên age mà dùng index trên age, và kết những bộ này lại. vì tập các bộ mà ta kết rất nhỏ nên chúng hợp với bộ nhớ và phép kết không quan trọng. dự định này giống như là thứ gì đó kém hơn so với dùng 1 chỉ mục trên dno, nhưng nó là reasonable alternative. Vì thế, nếu ta có 1 chỉ mục trên age rồi thì sự khác biệt của mẫu query này không khẳng định việc tạo chỉ mục trên trường dno của Employees.

Sau đây là ví dụ query loại range selection:

SELECT E.ename, D.dname

FROM Employees E, Departments D

WHERE E.sal BETWEEN 10000 AND 20000

AND E.hobby = ‘Stamps’ AND E.dno = D.dno

Câu truy vấn này minh hoạ việc dùng BETWEEN cho diễn tả phạm vi lựa chọn. nó tương đương với điều kiện sau:

10000 <= E.sal AND E.sal <= 20000

Quay lại ví dụ về câu truy vấn , cả 2 điều kiện lựa chọn đều trên quan hệ Employees. Vì thế, rõ ràng 1 dự định mà Employees là quan hệ ngoại và Departiments là quan hệ nội là tốt nhát, vì trong truy vấn trước, ta nên xây dựng 1 hash index trên thuộc tính dno của Departments. Nhưng chỉ mục nào nên được dùng trên Employees? 1 chỉ mục B+ trên thuộc tính sal sẽ giúp lựa chọn thuộc range selection, đặc biệt nếu nó là gom nhóm. 1 hash index trên thuộc tính hobby sẽ giúp lựa chọn loại equality selection. Nếu 1 trong số các chỉ mục có sẵn, ta có thể lấy các bộ của Employees dùng chỉ mục này, từ đó lấy các bộ Departments bằng cách dùng chỉ mục trên dno, và ứng dụng tất cả các lựa chọn còn lại và và các dự án nhanh lên. Nếu cả 2 chỉ mục đã sẵn sàng, người dùng thông minh sẽ chọn nhiều lựa chọn truy cập quỹ đạo để gửi các truy vấn; đó là, nó sẽ xét lựa chọn nào có ít các bộ thoả mãn hơn. Nhìn chung, cái nào truy cập quỹ đạo là có lựa chọn nhiều hơn phụ thuộc vào dữ liệu. nếu có rất ít người có salaries nằm trong phạm vi lựa chọn và nhiều người thu nhặt tem, thì dùng B+ là tốt nhất. cách khác, 1 hash index trên hobby là tốt nhất.

1. **gom nhóm và chỉ mục**

Range query là ứng cử tốt cho sự cải tiến với chỉ mục gom nhóm( clustered index):

SELECT E.dno

FROM Employees E

WHERE E.age >40

Nếu ta có 1 chỉ mục B+ trên age , ta có thể dùng nó để lấy các bộ mà thoả lệnh E.age > 40. 1 chỉ mục quan trọng phụ thuộc lệnh điều kiện lựa chọn mà đứng đâu tiên hay không, những người nào sẽ có tuổi hơn 40? Nếu trên thực tế tất cả mọi người đều lớn hơn 40 tuổi thì ta không phải nỗ lực nhiều trong việc tạo chỉ mục trong age; việc duyệt quan hệ hầu hết có thể tốt. tuy nhiên giả sử chỉ có 10% employees có tuổi lơn hơn 40 thì giờ chỉ mục có ích hay không? Câu trả lời phụ thuộc vào chỉ mục có được gom nhóm hay không, nếu chỉ mục không được gom nhóm thì ta có thể có 1 trang I/O cho mỗi employee thoả điều kiện, và điều này có thể rất tốn hơn là duyệt theo thứ tự thậm chí chỉ có 10% employees thoả điều kiện! mặt khác, 1 chỉ mục B+ gom nhóm trên age yêu cầu chỉ 10% I/)s cho 1 lần duyệt thứ tự.

Như 1 ví dụ khác, xét câu lệnh đươc sửa lại như sau:

SELECT E.dno, COUNT(\*)

FROM Employees E

WHERE E.age > 10

GROUP BY E.dno

Nếu 1 chỉ mục B+ có sẵn trên age, ta có thể lấy các bộ bằng cách dùng nó, sắp xếp các bộ nhận được theo dno, và sau đó trả lời câu truy vấn. tuy nhiên điều này có thể là 1 dự kiến không tốt nếu thực tế tất cả employees đều lớn hơn 10 tuổi, dự kiến này sẽ không hay nếu chỉ mục không được gom nhóm.

Ta hãy xét xem 1 chỉ mục trên dno có thể tiện lợi cho giả thiết hơn không, ta có thể dùng chỉ mục để nhận lấy tất cả các bộ, nhóm dno, và mỗi dno ta đếm số lượng bộ mà có age >10. Nhắc lại, hiệu quả phụ thuộc chủ yếu vào việc có nên dùng chỉ mục được gom nhóm hay không. Dự định này sẽ là tốt nhất nếu như không có điều kiện về tuổi. nếu 1 chỉ mục không có gom nhóm, ta có thể thực thi 1 trang I/O mỗi bộ trong Employees, và dự định này thật là kinh khủng, thực tế, nếu 1 chỉ mục không được gom nhóm, người dùng thông minh sẽ chọn dự định không phức tạp dựa vào sắp xếp theo dno. Vì thế, câu truy vấn này để nghị ta nên xây dựng 1 chỉ mục gom nhóm trên dno nếu điều kiện về age rất không có lựa chọn.nếu điều kiện rất có lựa chọn , ta nên tạo 1 chỉ mục trên age.

Việc gom nhóm cũng quan trọng cho 1 chỉ mục trên khoá tìm kiếm mà không chứa 1 khoá ứng củ, đó là 1 chỉ mục mà 1 vài thực thể có thể có cùng giá trị khoá trên nó. Để làm rõ vấn đề này, ta có truy vấn sau:

SELECT E.dno

FROM Employees E

WHERE E.hobby = ‘Stamps’

Nếu nhiều người sưu tầm tem, thì việc lấy các bộ trong chỉ mục trên hobby mà không có gom nhóm có thể rất không hiệu quả. Nó có thể đơn giản hơn duyệt quan hệ để lấy tất cả các bộ và để ứng dụng vào lựa chọn nhanh với các bộ nhận được. vì thế, nếu 1 truy vấn mà quan trọng thì ta nên xét đến tạo ra 1 chỉ mục trên hobby 1 chỉ mục gom nhóm. Mặt khác, nếu ta nhận rằng eid là 1 khoá của Employees, và thay thế điều kiện E.hobby = ‘Stamps’ bằng E.eid = 552, ta biết rằng hầu hết 1 bộ của 1 Employees sẽ đáp ứng điều kiện select. Trong trường hợp này, không có lợi gì khi làm chỉ mục gom nhóm.

Chỉ mục gom nhóm có thể đặc biệt quan trọng trong khi truy cập quan hệ nội bộ trong 1 chỉ mục được lồng vào kết lặp. để hiểu quan hệ giữa gom nhóm chỉ mục và phép kết, ta xem ví dụ sau:

SELECT E.ename, D.mgr

FROM Employees E, Departments D

WHERE D.dname = ‘Toy’ AND E.dno = D.dno

Ta kết luận rằng 1 dự định ước lượng tốt là dùng chỉ mục trên dname để lấy các bộ Departments thoả điều kiện trên dname và từ đó tìm bộ Employees bằng cách dùng chỉ mục trên dno. Ta có nên cho các chỉ mục này gom nhom không? Cho giả định của ta rằng 1 số lượng bộ thoả D.dname = ‘Toy’ dường như ít, ta nên xây dựng 1 chỉ mục không gom nhóm trên dname. Mặt khác, Employees là 1 quan hệ nội bộ trong 1 chỉ mục được lồng vào phép kết lặp, và dno thì không là khoá . tình huống này là 1 lý lẽ mạnh cho biết dùng chỉ mục gom nhóm trên dno.

Xét ví dụ sau:

SELECT E.ename, D.mgr

FROM Employees E, Departments D

WHERE E.hobby = ‘Stamps’ AND E.dno = D.dno

Trong trường hợp này, 1 block nested loops hay soet-merge kết có thể gây nhiều rắc rối. 1 kết sort-merge có thể lấy lợi ích của 1 chỉ mục B+ trên dno của Departments để lấy các bộ và tránh việc sắp xếp Departments. Chú ý là 1 chỉ mục không gom nhóm thì không có ích – vì tất cả bộ được nhận, việc thực thi 1 I/O mỗi bộ dường như chi phí cao. Nếu không có chỉ mục trên dno của Employees, ta có thể nhận bộ Employees, ứng dụng Selection E.hobby = ‘Stamps’ nhanh, và xếp các bộ thoả theo dno.

* + 1. ***Co-clustering Two Relations***

**Tóm tắt về co-clustering:**

* Nó có thể làm tăng tốc độ phép kết, đặc biệt khoá ngoại kết tương ướng với quan hệ 1:N.
* Duyệt tuần tự của quan hệ sẽ làm chậm hơn.
* Lệnh insert, delete, update thay đổi độ dài dữ liệu sẽ trở nên chậm đi, nhờ các chi phí liên quan đến việc duy trì các gom nhóm.

1. **chỉ mục trên khoá tìm kiếm mà nhiều thuộc tính (indexes on multiple-attribute search keys**

Thỉnh thoảng sẽ rất tốt nếu xây dựng 1 chỉ mục trên khoá tim kiếm mà chứa nhiều thuộc tính.ví dụ, nếu ta muốn lấy các dữ liệu trong Employees với điều kiện age = 30 và sal = 4000, thì 1 chỉ mục với khoá tìm kiếm là (age, sal) (hoặc sal, age) thì tốt hơn chỉ mục với khoá tìm kiếm là age hoăc là sal. Nếu có 2 chỉ mục, 1 là trên age và 1 là tên sal, ta có thể dùng chúng để thực hiện câu truy vấn bằng cách lấy và giao nhau. Tuy nhiên nếu ta xét đến những chỉ mục nào mà được tạo ra vì mục địch như câu truy vấn này thì dùng chỉ mục đa thuộc tính thì tốt hơn.

Composite index có thể dùng đề phục vụ cho truy vấn phạm vi nhiều chiều.

Ví dụ:

SELECT E.eid

FROM Employees E

WHERE E.age BETWEEN 20 AND 30 AND E.sal BETWEEN 3000 AND 5000

Composite index trên (age, sal) giúp khi có yêu cầu trong lệnh WHERE có sự lựa chọn 1 cách rõ ràng. Một cách rõ ràng, 1 hash index không thể giúp , 1 B+ index được yêu cầu làm, khi ta thấy 1 chỉ mục gom nhóm sẽ giúp cải tiến hơn chỉ mục không gom nhóm. Trong ví dụ này ta thấy với chỉ mục B+ có khoá tìm kiếm là (age, sal) thì cũng như chỉ mục B+ có khoá là (sal, age). Tuy nhiên thứ tự các thuộc tính trong khoá tìm kiếm đôi khi cũng có nhiều sai khác lớn, như query dưới đây:

SELECT E.eid

FROM Employees E

WHERE E.age = 25

AND E.sal BETWEEN 3000 AND 5000

Đây là câu lệnh có chỉ mục composite index với chỉ mục B+ gom nhóm, trên (age, sal), điều này cho thực thi của câu lệnh tốt vì dữ liệu được sắp xếp bởi age và sau đó theo sal. Như vậy, tất cả các dử liệu nào có age = 25 thì cùng gom 1 nhóm. Mặt khác, 1 chỉ mục hỗn hợp B+ có gom nhóm khoá trên(sal, age) sẽ cho thực thi không tốt. vì dữ liệu hàng sẽ được xếp theo sal trước và khi đó 2 hàng có cùng giá trị age có thể nằm ở xa nhau. Kết quả, chỉ mục này cho phép ta dùng lựa chọn phạm vi cho sal nhưng không có lựa chon cho age để lấy các bộ ra.

Mặt trái của composite index: phải được update đáp lại cho bất kỳ thao tác nào mà làm chỉnh sửa(insert, delete, update) đến các thuộc tình mà tham gia làm khoá tìm kiếm. 1 composite index dường như lớn hơn chỉ mục có khoá là 1 thuộc tính vì kích thước của các entry lớn hơn.

1. **Chỉ mục cho phép những dự án chỉ có chỉ mục(indexes that enable index-only plans)**

Mục này xét về 1 lượng lớn truy vấn mà ta có thể tìm những dự định mà mang lại hiệu quả để tránh lấy ra các bộ tự 1 trong những quan hệ mà phải liên kết đến(referenced) cho chúng. 1 chỉ mục chỉ được dùng để duyệt mà không phải gom nhóm vì những bộ mà từ quan hệ được chỉ mục thì không được lấy ra! Tuy nhiên, chỉ có chỉ mục chằn chịt(dense) mới có thể được dùng cho index-only.

Ví dụ:

SELECT D.mgr

FROM Departments D, Employees E

WHERE D.dno = E.dno

Quan sát thấy không có thuộc tính nào của Employees được lấy ra. Nếu ta có dense index trên trường dno của Employees, sự tối ưu hoá của việc tạo 1 chỉ mục được lồng vào kết lặp mà sử dụng 1 index-only để duyệt quan hệ nội bộ thì thích hợp.

1. **xem qua tinh chỉnh dữ liệu (overview of database tuning)**

sau khi thiết kế dữ liệu mức vật lý thì ta có bước tình chỉnh dữ liệu, vì trong quá trình thiết kế sẽ khó tránh khỏi nhưng sai xót, do đó bước tình chỉnh sẽ là bước hoàn thiện dữ liệu mức vật lý. Tinh chỉnh dữ liệu giúp cho khả năng tăng tốc thực thi cao hơn. Trong phần này ta bàn về tinh chỉnh chỉ mục, tinh chỉnh lược đồ quan hệ, tinh chỉnh câu truy vấn.

* + 1. ***tuning indexs***

Việc lựa chọn chỉ mục ban đầu cũng có thể được tinh chỉnh lại do 1 số lý do. Lý do đơn giản nhất là dựa vào khối lượng công việc quan sát cho thấy rằng 1 số các truy vấn và cập nhậ được coi là quan trọng trong đặc tả khối lượng công việc ban đầu không phải là rất thường xuyên. Và khối lượng công việc quan sát cũng cho ra những truy vấn quan trọng nhưng trong đặc tả của ta ban đâu không có. Vì thế 1 số chỉ mục sẽ cần được tạo ra hoặc bị gỡ bỏ. lúc này ta sẽ thực hiện tinh chỉnh chỉ mục.

* + 1. ***tuning the conceptual schema***

Như ta biết, trong thiết kế lược đồ, lược đồ ở mức quan niềm mà ta lựa chọn chưa hẳn là lược đồ hợp lý so với thực tế (workload) yêu cầu mặc dù ở mức thiết kế vật lý làm hết sực khả năng có thể. Vì vậy ta có thể phải điều chỉnh lại lược đồ mức quan niệm của ta cho phù hợp với workload. Một vài điều phải xét đến khi tinh chỉnh lược đồ quan hệ;

* + Ta cần quyết định chọn dạng chuẩn thiết kề nào: 3NF hay BCNF.
  + Nếu có 2 cách để phân rã cho ra lược đồ từ dạng chuẩn 3NF hay BCNF , thì ta cần dựa vào workload để lựa chọn lược đồ nào.
  + Thỉnh thoảng ta nên quyết định phân rã quan hệ nữa không mặc dù nó đã đạt BCNF.
  + Trong 1 số tình huống ta có thể không theo tiêu chuẩn hoá. Đó là, ta cần phải lựa chọn để loại bỏ 1 quan hệ mà có được do phân rã từ quan hệ lớn hơn

## *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.
* 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. 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. Nhưng để làm được như vậy, người dùng phải thấu hiểu cả bộ tối ưu và khả năng của DBMS

# 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))*

Ta ký hiệu các thuộc tính bằng ký tự đơn, và lược đồ quan hệ kí hiệu bằng dãy các kí tự. Xem xét lược đồ quan hệ Contracts, ta 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

## *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 cách 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.

## *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.

## *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

## *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.

## *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.

# 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 đáng để làm là trước khi tạo index mới chúng ta nên xem xét viết lại truy vấn có thể đạt kết quả chấp nhận được với index hiện tại hay không.

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.

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.

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.

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.

# 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.

# 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)

## *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.

## *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