TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Thiết kế quản trị cơ sở dữ liệu

Đề tài: Using Bind Variables

Giảng viên hướng dẫn T.s Trần Việt Trung

Họ & Tên	MSSV	Lớp
Lê Thành Đạt	20111374	CNTT2.2-K56
Nguyễn Văn Cương	20111205	CNTT2.2-K56
Nguyễn Đức Sơn	20112069	CNTT2.2-K56
Phạm Thế Quyền	20112050	CNTT2.2-k56

Hà Nội 5 – 2015

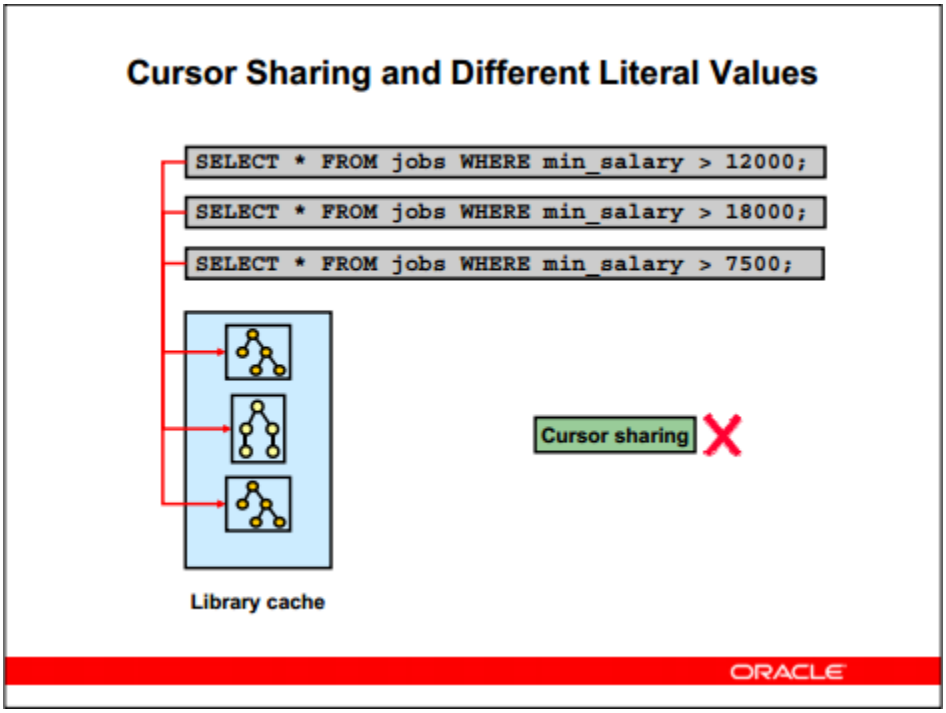
Mục Lục

I. Chia sẻ con trỏ và các giá trị điều kiện khác nhau

Nếu ta sử dụng câu truy vấn SQL mà có sử dụng các giá trị điều kiện trong mệnh đề Where sẽ có nhiều câu truy vấn gần như giống hệt nhau và đều được lưu trữ trong bộ nhớ cache. Với mỗi câu truy vấn, bộ tối ưu phải thực hiện toàn bộ các bước như 1 câu truy vấn mới. Đó cũng là lý do bộ nhớ cache nhanh tràn bởi tất cả các câu truy vấn khác nhau đều được lưu trữ tại đây.

Khi ta thực hiện như vậy, ta sẽ không sử dụng được lợi ích từ việc chia sẻ con trỏ. Cho nên, để con trỏ được chia sẻ, ta nên sử dụng biến ràng buộc hơn là sử dụng các giá trị điều kiện, mỗi con trỏ được chia sẻ gắn với một kế hoạch thực thi riêng. Tuy nhiên tùy thuộc vào các giá trị điều kiện được cung cấp sẽ có những kế hoạch thực thi khác nhau, chúng được sinh bởi bộ tối ưu.

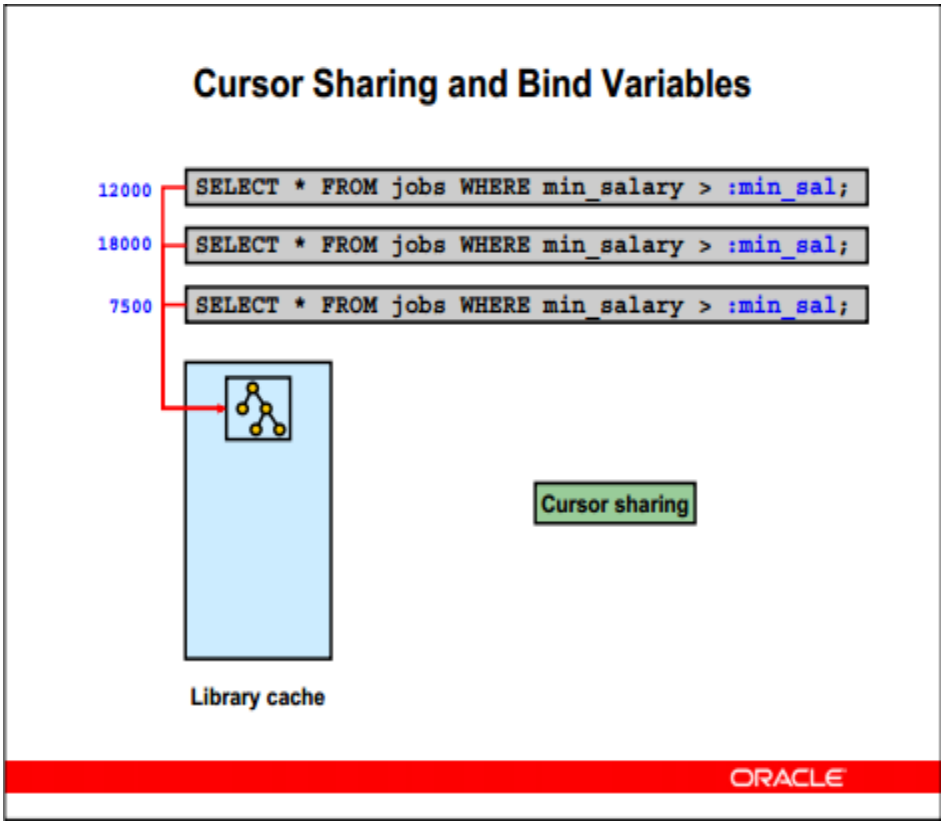
Ví dụ như, ta có thể thu được một số kết quả trả về của bảng JOBS khi mà điều kiện min_saraly >12000, hoặc môt vài kết quả trả về khi min_saraly > 18000. Sự khác biệt ở đây là sự phân bố của dữ liệu, do đó các kế hoạch thực thi khác nhau có thể được sử dụng tùy thuộc vào giá trị đặt trong các câu truy vấn.



Nhìn vào hình trên ta có thể thấy, câu truy vấn đầu tiên và thứ ba có cùng một kế hoạch thực hiện, còn câu thứ hai thì khác biệt hoàn toàn. Nếu xét theo quan điểm về mặt hiệu năng, việc có con trỏ riêng sẽ tốt hơn. Tuy nhiên, nó không được tiết kiệm lắm bởi vì ta có thể chia sẻ con trỏ cho hai câu truy vấn một và ba như trong ví dụ trên.

II. Chia sẻ con trỏ và biến ràng buộc

Thay vì sử dụng các câu truy vấn khác nhau cho mỗi giá trị điều kiện, ta có thể sử dụng một biến ràng buộc, khi đó, các hoạt động phân tích thêm sẽ bị loại bỏ(về mặt lý thuyết). Đó là bởi vì bộ tối ưu có thể nhận ra rằng câu truy vấn đã được phân tích và quyết định sử dụng lại kế hoạch thực thi, mặc dù ta có gán các giá trị khác nhau cho biến đó ở các lần tiếp theo.



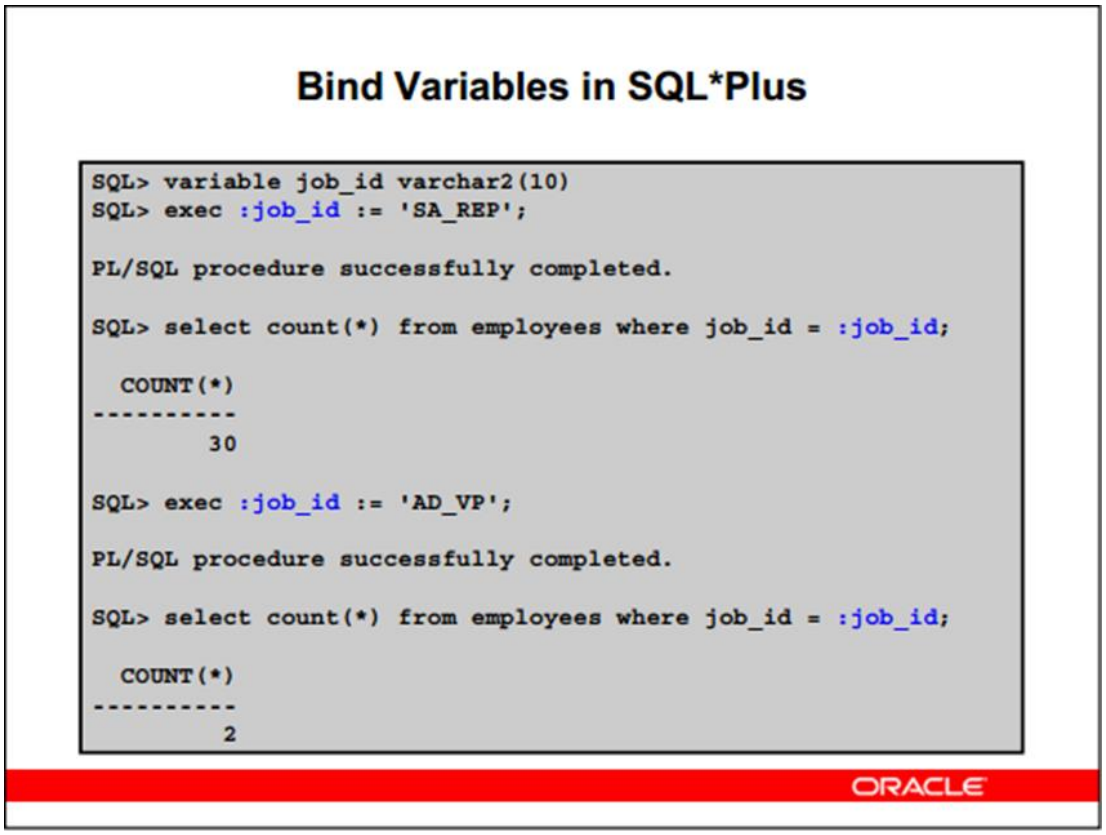
Trong ví dụ trên, biến ràng buộc là min_sal. Nó được so sánh với giá trị của cột min_salary trong bảng JOBS. Thay vì đưa ra 3 câu truy vấn khác nhau, ta chỉ sử dụng một câu truy vấn với biến ràng buộc. Tại thời điểm thực thi, các kế hoạch thực thi giống nhau sẽ được sử dụng, các giá trị đưa vào sẽ thay thế cho biến.

Tuy nhiên, cũng dựa trên quan điểm về mặt hiệu năng, đây không phải là cách giải quyết tốt nhất. Như ví dụ trên, ta chỉ có thể có được hiệu năng tốt

nhất với hai câu truy vấn 1 và 3, còn câu truy vấn số 2 thì không. Nhưng mặt khác, nó rất tiết kiệm bởi vì ta chỉ cần một con trỏ chia sẻ trong bộ nhớ thư viện cache để thực thi tất cả 3 câu truy vấn. Sau đây ta sẽ xem xét một vài ví dụ về việc sử dụng biến ràng buộc trong SQL.

III. Biến ràng buộc trong truy vấn SQL

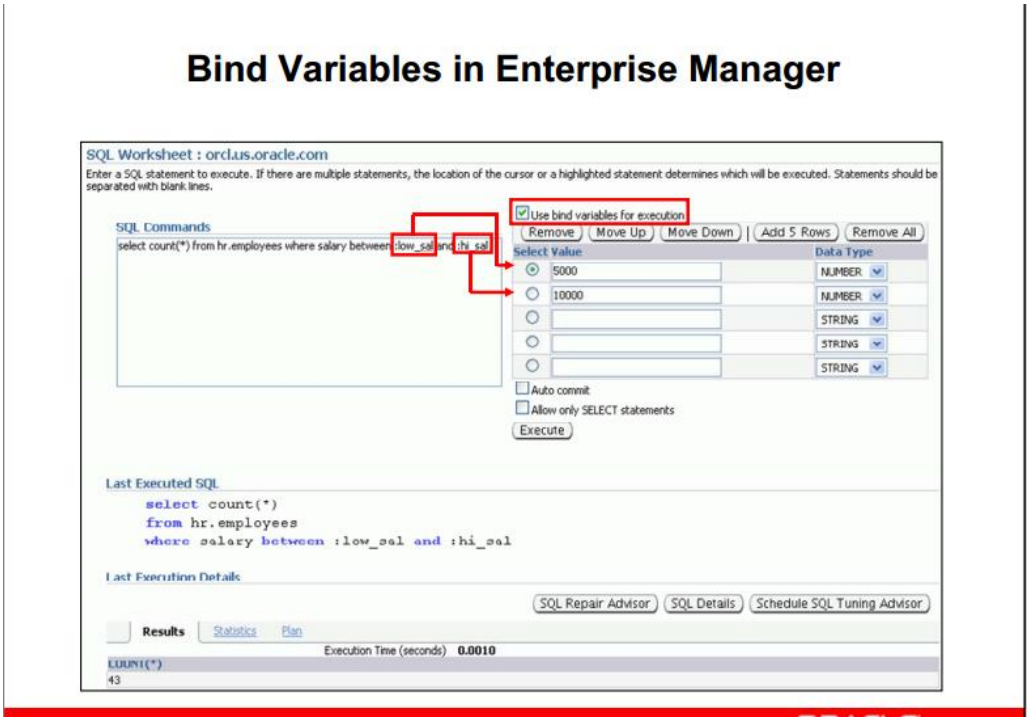
3.1. Biến ràng buộc trong SQL * Plus



Biến ràng buộc có thể sử dụng trong SQL* Plus. Trong SQL* Plush , sử dụng lệnh VARIABLE để khởi tạo một biến ràng buộc. Sau đó, ta có thể gán giá trị cho biến bằng câu lệnh EXEC. Kể từ đó, bất kì xử lý nào với biến đều tương ứng với giá trị đã gán.

Trong ví dụ trên, Khi gán giá trị SA_REP cho biến job_id thì kết quả count(*) trả về là 30, sau đó khi gán giá trị AD_VP cho biến thì kết quả trả về là 2.

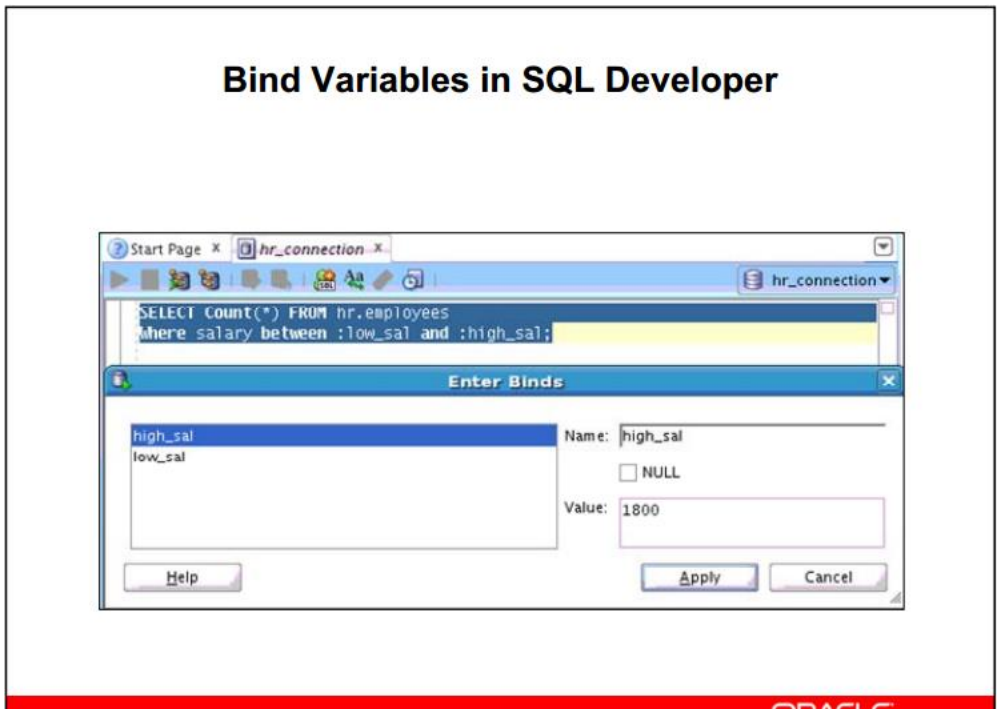
3.2. Biến ràng buộc trong Enterprise Manager



Trên trang SQL Worksheet của Enterprise Manager, ta có thể thấy rõ câu lệnh SQL đã sử dụng biến ràng buộc. Ta có thể làm điều đó bằng cách chọn vào ô "Use bind variables for execution" nghĩa là "Sử dụng biến ràng buộc để thực thi". Khi ta lựa chọn như vậy, một vài trường sẽ được sinh ra, nơi mà ta có thể đặt giá trị cho các biến ràng buộc.

Hãy tham khảo cách đặt biến ràng buộc trong câu lệnh SQL bắt đầu với dấu hai chấm ":". Thứ tự gán giá trị cũng chính là thứ tự khai báo biến ràng buộc trong câu lệnh SQL(nhìn hình vẽ trên ta có thể thấy rõ điều đó). Biến đầu tiên được gán ứng với giá trị đầu tiên, biến thứ hai được gán ứng với giá trị thứ hai, và cứ tiếp tục như vậy. Nếu ta thay đổi giá trị các biến trong câu lệnh SQL, ta có lẽ lên thay đổi lại danh sách giá trị cho phù hợp.

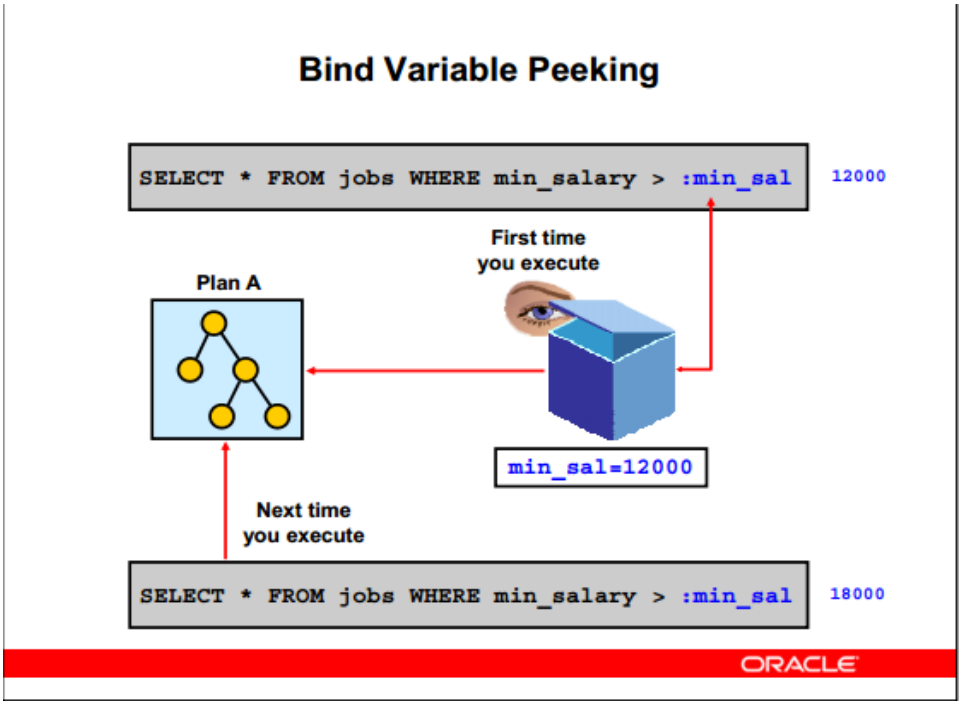
3.3. Biến ràng buộc trong SQL Developer



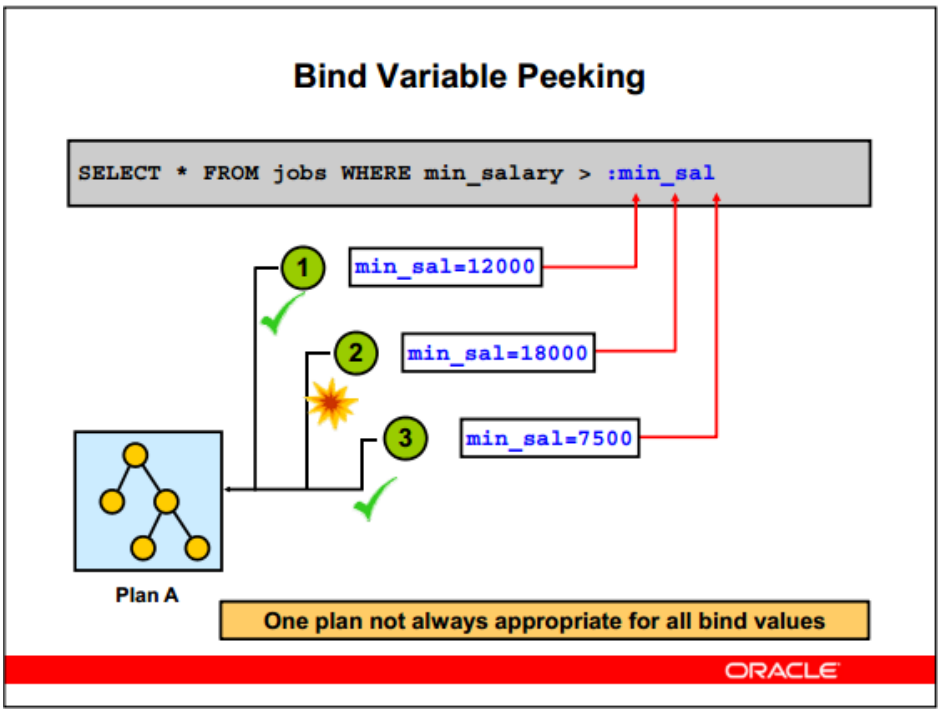
Trong khung làm việc của SQL, ta có thể chỉ rõ biến ràng buộc được sử dụng. Khi ta thực thi câu truy vấn, một hộp thoại Enter Binds sẽ hiện ra, như trên hình, khi đó ta có thể đặt giá trị cho biến ràng buộc.

Tham khảo giá trị trong câu lệnh SQL được sử dụng bởi biến ràng buộc bắt đầu bởi dấu ":". Chọn từng biến để có thể thay đổi giá trị cho chúng.

IV. Nhìn trộm giá trị biến ràng buộc



Khi một giá trị điều kiện được sử dụng, giá trị đó được sử dụng bởi bộ tối ưu nhằm quyết định kế hoạch thực thi tối ưu nhất. Nhưng nếu sử dụng biến ràng buộc thì hệ tối ưu vẫn cần chọn ra kế hoạch tốt nhất dựa trên các giá trị trong mệnh đề điều kiện của câu truy vấn, nhưng ta không thể xác định được giá trị trong dạng văn bản của câu truy vấn. Điều đó có nghĩa là khi một câu truy vấn được phân tích, hệ thống cần có khả năng thấy được giá trị của biến ràng buộc, để chắc chắn rằng kế hoạch thực thi là phù hợp với giá trị đó. Bộ tối ưu làm việc đó dựa vào việc nhìn trộm (peeking) giá trị của biến ràng buộc. Với các câu truy vấn mà việc phân giải phức tạp thì bộ tối ưu sẽ đánh giá từng biến ràng buộc rồi từ đó sử dụng chúng làm đầu vào để chọn ra phương án thực thi tốt nhất. Sau khi việc thực thi được xác định trong lần phân tích đầu tiên, nó sẽ được tái sử dụng cho các câu truy vấn tương tự mà không phụ thuộc và giá trị của biến ràng buộc.



Dưới một vài điều kiện, việc lén nhìn giá trị của biến có thể làm bộ tối ưu lựa chọn được kế hoạch thực thi tốt nhất. Điều đó xảy ra bởi vì giá trị đầu tiên của biến được sử dụng để xác định kế hoạch cho các lần thực thi sau đó. Bởi vậy, mặc dù các lần thực thi khác nhau được cung cấp các giá trị của biến khác nhau nhưng sẽ có cùng một kế hoạch thực thi. Và có thể đối với vài giá trị ràng buộc sau có 1 kế hoạch thực thi tốt hơn cho câu truy vấn đó. Một ví dụ như nơi lựa chọn là các chỉ mục đặc thù. Đối với sự chọn lựa mức thấp thì việc duyệt qua toàn bộ bảng còn nhanh hơn. Đối với sự lựa chọn mức cao, duyệt trên cây chỉ mục là thích hợp hơn. Với ví dụ, kế hoạch A là tốt với câu truy vấn 1 và 3 nhưng với câu thứ 2 thì không vậy. Cho rằng các kết quả trả về là rất ít khi điều kiện là min_salary > 18000, và kế hoạch A là duyệt toàn bộ bảng. Trong trường hợp này, duyệt qua toàn bảng là không hề tốt với câu truy vấn thứ 2.

Biến ràng buộc rất có lợi, chúng tạo ra nhiều con trỏ chia sẻ do đó giảm phân tích cú pháp SQl. Như trong vi dụ nó có thể tạo ra kế hoạch tối ưu cho vài giá trị của biến rằng buộc mà không phải toàn bộ, điều này giải thích tại sao không sử dụng biến ràng buộc cho hệ trợ giúp quyết định (DSS). Vì ở đây việc phân tích cú pháp chỉ mất phần trăm giây trong khi việc thực thi có thể hàng phút đến hàng giờ. Việc thực thi với 1 kế hoach chậm hơn là không đạt được giá trị tiết kiệm trong thời gian phân tích cú pháp.

V. Đẩy mạnh việc chia sẻ con trỏ

Oracle 8i giới thiệu rằng khả năng chia sẻ con trỏ của các câu truy vấn khác nhau chỉ trên các giá trị điều kiện. Thay vì đi xây dựng một kế hoạch thực thi với mỗi câu truy vấn giống nhau chỉ khác các giá trị điều kiện đầu vào, bộ tối ưu sẽ sinh ra một kế hoạch thực thi chung cho tất cả lần thực thi sau đó. Bởi vì chỉ một kế hoạch được sử dụng thay vì sử dụng các kế hoạch thực thi khác nhau cho mỗi câu truy vấn, tính năng này có thể được kiểm tra lại trên ứng dụng của ta trước khi quyết định có dung nó hay không. Đó là lý do tại sao Oracle 9i mở rộng tính năng này bằng cách chia sẻ con trỏ với các câu truy vấn được coi là giống nhau.Đó là chỉ khi bộ tối ưu đảm bảo rằng kế hoạch thực thi là không phụ thuộc vào các giá trị điều kiện được sử dụng.Ví dụ, hãy xem xét câu truy vấn sau, khi mà EMPLOYEE_ID là khóa chính:

```
SQL> SELECT * FROM employees
  2  WHERE employee_id = 153;
```

Chúng ta có thể thấy rằng mọi giá trị của trường EMPLOYEE_ID đều có vai trò như nhau(duy nhất một giá trị) nên sẽ có một kế hoạch thực thi là như nhau.Chính vì thể, để cho tiết kiệm bộ tối ưu sẽ sinh ra chỉ một kế hoạch chung cho tất cả các lần thực thi khi mà thay giá trị employee_id khác nhau.

Mặt khác, cho rằng cùng với bảng EMPLOYEES đó, có một trường thuộc tính là DEPARTMENT_ID có một khoảng giá trị rộng. Ví dụ , với department 50 có thể chiếm tới hơn một phần ba của bảng employees, trong khi đó với giá trị department 70 chỉ có một hai giá trị.

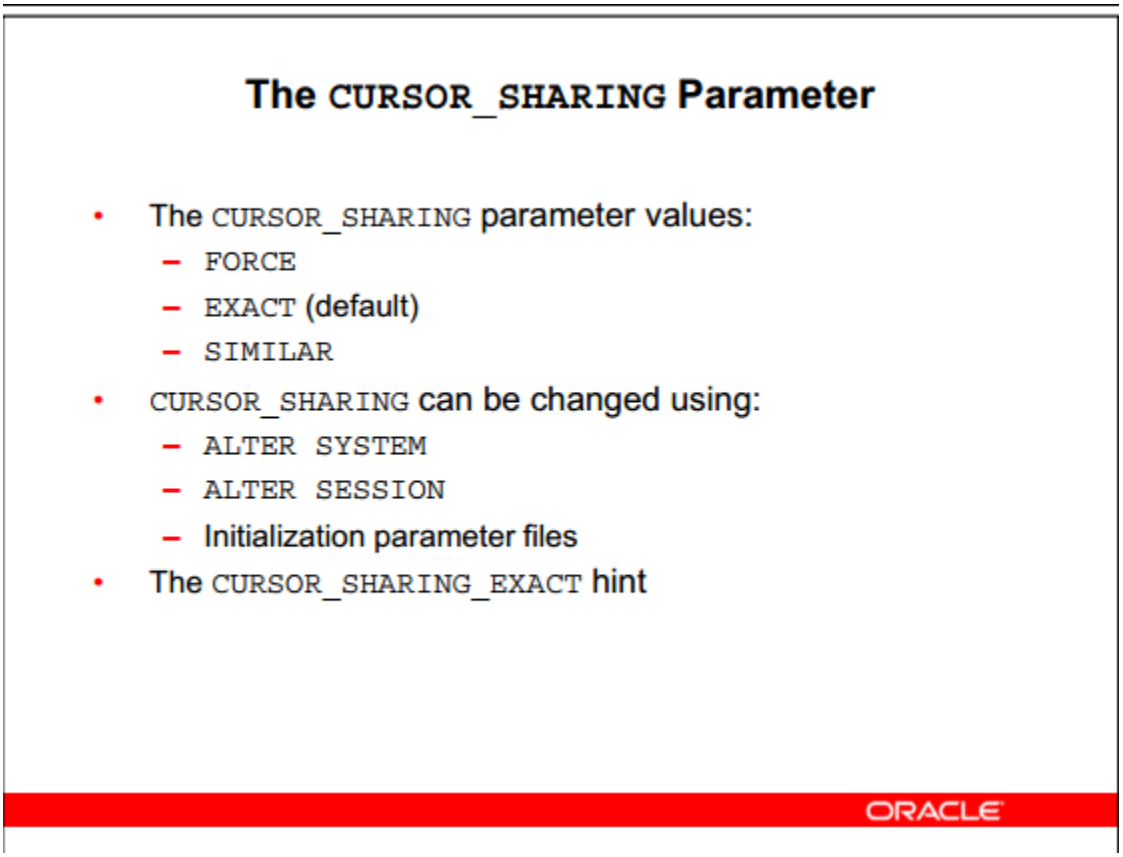
Xét hai câu truy vấn sau :

```
SQL> SELECT * FROM employees
  2  WHERE department_id = 50;
```

```
SQL> SELECT * FROM employees
  2  WHERE department_id = 70 ;
```

Có thể thấy ở đây, nếu sử dụng việc chia sẻ con trỏ, thực sự là không tiết kiệm tý nào khi mà ta đã có sự thống kê tần suất các giá trị của cột department_id. Trong trường hợp này, phụ thuộc vào câu truy vấn nào thực hiện trước, kế hoạch thực thi có thể bao gồm duyệt qua toàn bộ bảng, hoặc duyệt qua khoảng chỉ mục đơn giản.

VI. Tham số CURSOR_SHARING



Giá trị của tham số CURSOR_SHARING được khởi tạo để xác định cách mà bộ tối ưu thực thi câu truy vấn với biến ràng buộc.

EXACT: việc thay thế giá trị điều kiện bị vô hiệu hóa hoàn toàn.
FORCE: Chia sẻ nguyên nhân cho mọi giá trị điều kiện.
SIMILAR: Chỉ chia sẻ nguyên nhân cho các giá trị điều kiện an toàn.

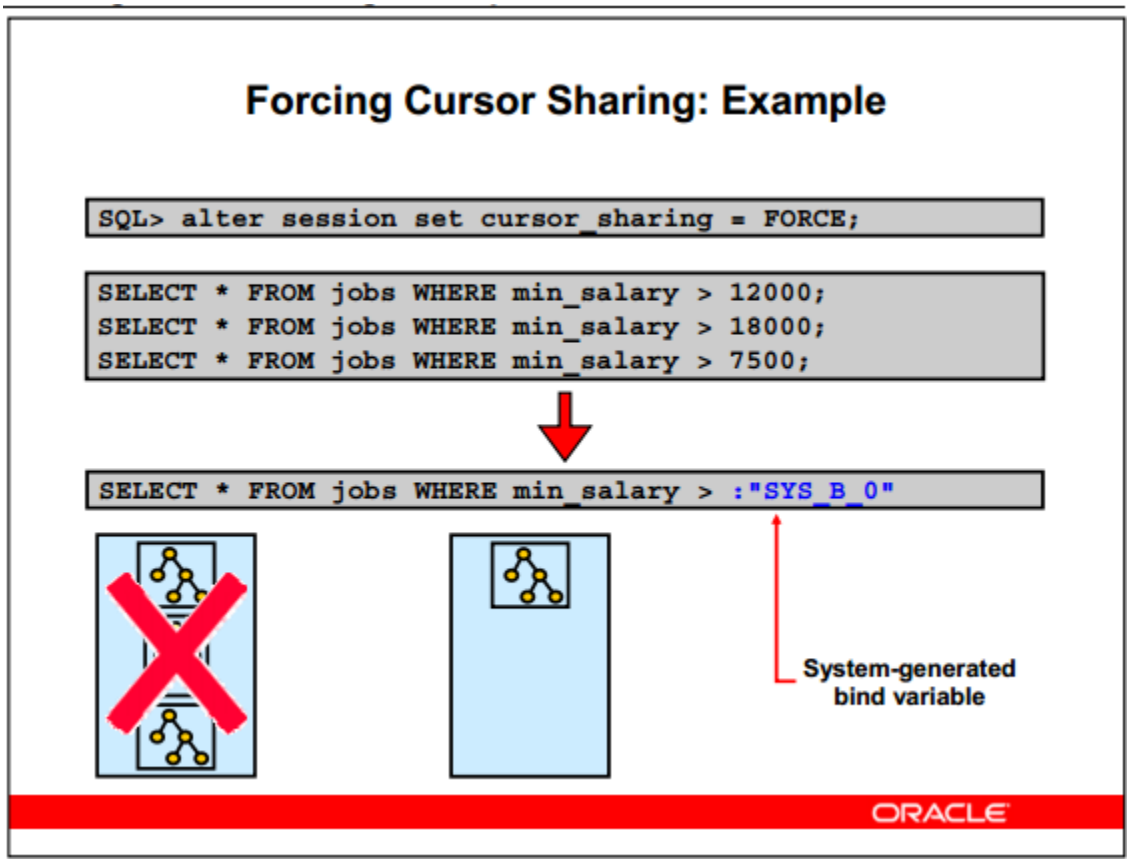
Trong những phiên bản trước đó, ta chỉ có hai lựa chọn là EXACT và FORCE. Lựa chọn Similar được bộ tối ưu sử dụng để kiểm tra câu truy vấn và đảm bảo rằng sự thay thế chỉ diễn ra với các giá trị điều kiện an toàn.Nó có thể tự nhiên sử dụng thông tin về các chỉ mục (unique hoặc nounique) và thống kê thu thập trên các chỉ mục hoặc các bảng dưới,bao gồm cả tần suất.

Giá trị của CURSOR_SHARING trong tệp khởi tạo có thể nạp lại bằng ALTER SYSTEM SET CURSOR_SHARING hoặc ALTER SESSION SET CURSOR_SHARING.

CURSOR_SHARING_EXACT dùng để nhắc nhở hệ thống thực thi câu truy vấn không được thay thế giá trị điều kiện bằng biến ràng buộc.

VII. Một số phương pháp chia sẻ con trỏ

7.1. Chia sẻ con trỏ bắt buộc



Ta có thể chia sẻ con trỏ bắt buộc bằng cách dùng câu lệnh ALTER SESSION, khi đó tất cả câu truy vấn của ta, dù khác nhau giá trị điều kiện hệ thống sẽ tự động sinh ra biến ràng buộc được gọi là SYS_B_0 như trong ví dụ trên. Như kết quả, ta sẽ kết thúc với chỉ một con trỏ thay vì ba.

7.2. Chia sẻ con trỏ thích nghi

7.2.1. Tổng quan chia sẻ con trỏ thích nghi

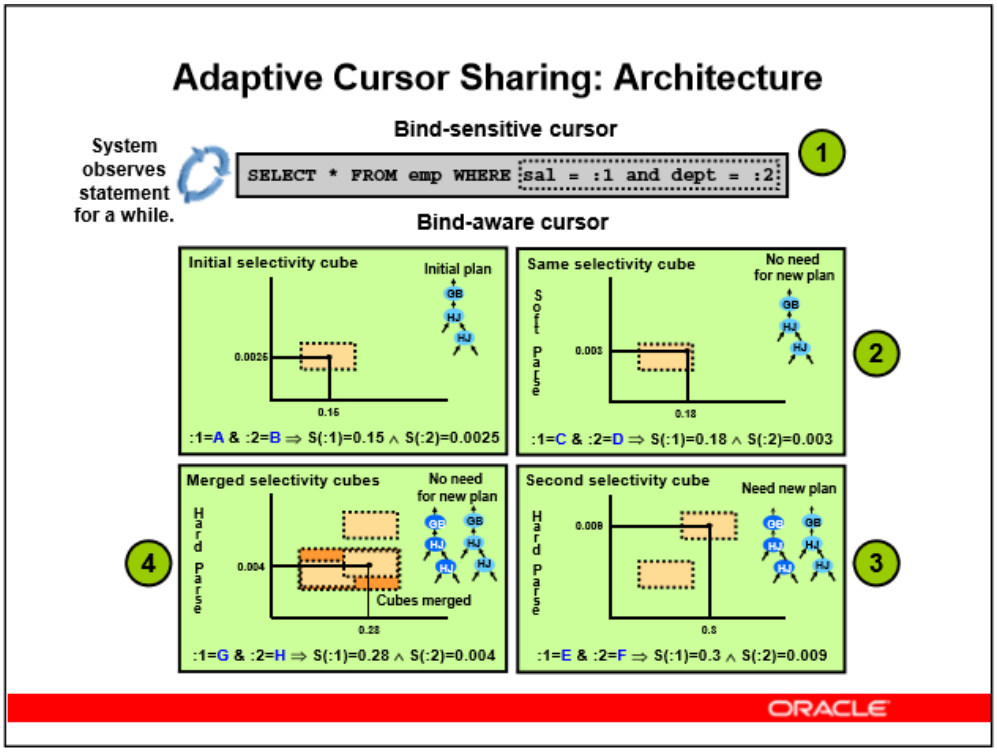
Biến ràng buộc được thiết kế cho phép database Oracle chia sẻ một con trỏ cho nhiều câu truy vấn SQL để giảm dung lượng bộ nhớ chia sẻ được sử dụng để phân giải câu truy vấn. Tuy nhiên chia sẻ con trỏ và bộ tối ưu sẽ xung đột về mục đích, việc viết 1 câu truy vấn với giá trị điều kiện sẽ cung

cấp nhiều thông tin hơn cho bộ tối ưu và đương nhiên dẫn đến chọn được kế hoạch thực thi tốt hơn, trong khi sẽ tăng bộ nhớ và chi phí cho CPU vì phải phân tích quá nhiều.

Trong phiên bản Oracle9i phiên bản đầu tiên giới thiệu về giải pháp thỏa hiệp dựa trên việc cho phép những câu truy vấn SQL tương tự sau sử dụng giá trị điều kiện khác nhau được chia sẻ . Đối với các truy vấn sử dụng biến ràng buộc Oracle9i cũng giới thiệu khái niệm rằng buộc nhìn trộm(bind peeking) . Để hưởng lợi từ ràng buộc nhìn trộm , hệ thống sẽ giả định chia sẻ con trỏ được dự định chia sẻ và những lời gọi khác nhau của câu truy vấn cũng được giả sử sử dụng các kế hoạch truy vấn tương tự.Nếu lời gọi khác của câu truy vấn hưởng lợi đáng kể từ kế hoạch thực thi khác thì ràng buộc nhìn trộm không thể tạo ra một kế hoạch thực thi tốt.

Giải quyết vấn đề càng nhiều càng tốt, Oracle11g giới thiệu chia sẻ con trỏ thích ứng, tính năng này là chiến lược tinh vi hơn , nó được thiết kế để không chia sẻ con trỏ một cách mù quáng nhưng tạo ra nhiều kế hoạch cho 1 câu truy vấn với biến ràng buộc nếu lơi ích mang lại từ việc dùng nhiều kế hoạch thực thi lớn hơn trong thời gian phân tích và chi phí sử dụng bộ nhớ.Tuy nhiên vì mục đích sử dụng biến ràng buộc là chia sẻ bộ nhớ nên sự thỏa hiệp cần tìm ra mối liên hệ với số con trỏ

7.2.2. Kiến trúc chia sẻ con trỏ thích nghi



Khi ta sử dụng Adaptive Cursor Sharing, các bước sẽ diễn ra được minh họa trong hình vẽ.

- Con trỏ bắt đầu vòng đời của nó bằng một phân tích cứng như thường lệ. Nếu ràng buộc nhìn trộm diễn ra, và một biểu đồ được sử dụng để tính toán sự chọn lọc của các vị từ mà chứa các biến ràng buộc, sau đó con trỏ được đánh dấu như một con trỏ ràng buộc nhạy cảm. Ngoài ra, một số thông tin được lưu trữ về các vị từ có chứa các biến ràng buộc, bao gồm cả tiêu chí chọn lọc. Trong ví dụ, tiêu chí chọn lọc đó sẽ được lưu trữ là một khối lập phương trung tâm xung quanh (0.15,0.0025). Bởi ban đầu là phân tích cứng, một kế hoạch thực hiện ban đầu được xác định bằng cách sử dụng với giá trị nhìn trộm. Sau khi con trỏ được thực thi, các giá trị ràng buộc và các số liệu thống kê thực hiện của con trỏ được lưu trong con trỏ. Trong việc thực hiện tiếp theo, khi có một tập mới giá trị điều kiện, hệ thống sẽ sử dụng phân tích mềm và tìm một con trỏ kết quả cho phân tích này. Vào lúc cuối việc thực hiện, thì các thống kê được so sánh với các giá trị đã lưu trong con trỏ. Và hệ thống sẽ so sánh các số liệu

thống kê trong các lần chạy trước đó và quyết định xem có đánh dấu con trỏ này là một ràng buộc đã biết hay không.

- Trong bước tiếp theo phân tích mềm của câu truy vấn, nếu con trỏ là một ràng buộc đã biết thì kết quả con trỏ đã biết này sẽ được sử dụng. Giả sử bây giờ vị từ được tạo giá trị ràng buộc là ((0.18,0.003). Bởi vì sự chọn đã được sử dụng như kết quả cũ, và sự lựa chọn này đã tồn tại trong khối lập phương đã có, nên ta sẽ sử dụng kế hoạch thực hiện của con trỏ con đã biết.

- Giả sử bây giờ vị từ được tạo giá trị ràng buộc là (0.3,0.009), bởi vì sự chọn lọc này thì ko nằm trong khối lập phương hiện tại, nên không kết quả con trỏ con nào được tìm thấy. Vì vậy hệ thống sẽ phải sử dụng phân tích cứng, điều này sẽ tạo một con trỏ con mới với một kế hoạch thực hiện khác. Ngoài ra, hình lập phương biểu diễn sự chọn lọc này cũng sẽ được lưu lại như một phần của con trỏ mới. sau khi thực hiện xong thì hệ thông lưu lại các giá trị ràng buộc và các thống kê việc thực hiện, lưu lại trong con trỏ.

- Bây giờ , trên một phân tích mềm tiếp theo, giả sử giá trị ràng buộc bây giờ là (0.28,0.004). Bởi vì sự chọn lọc này thì ko nằm trong khối lập phương hiện tại, nên hệ thống sẽ phải sử dụng phân tích cứng. Tuy nhiên thì kế hoạch lại giống cái đầu tiên, nên cả 2 con trỏ con này sẽ được hợp lại. Cả 2 hình lập phương của mỗi con trỏ con sẽ được sáp nhập vào một khối lớn. Còn một con trỏ con sẽ được xóa đi. Lần sau nếu sự lựa chọn mà nằm trong khối mới này thì sẽ có con trỏ phù hợp.

7.2.3. Chi tiết con trỏ chia sẻ thích nghi

Những quan điểm xác định xem một truy vấn được ràng buộc ý thức hay không, và được xử lý tự động, mà không có bất kỳ tác động của người sử dụng. Tuy nhiên, thông tin về những gì diễn ra được giử vào V$ để ta có thể chuẩn đoán vấn đề nếu nó phát sinh.Cột mới đã được thêm vào V$ SQL:

- IS_BIND_SENSITIVE: Chỉ ra nếu một con trỏ được ràng buộc nhạy cảm; giá trị YES|NO. Câu truy vấn mà bộ tối ưu lén nhìn các giá trị biến ràng buộc khi tính toán tiêu chí chọn lọc và một sự thay đổi giá trị biến ràng buộc có thể dẫn đến một kế hoạch khác nhau được gọi là ràng buộc nhạy cảm.

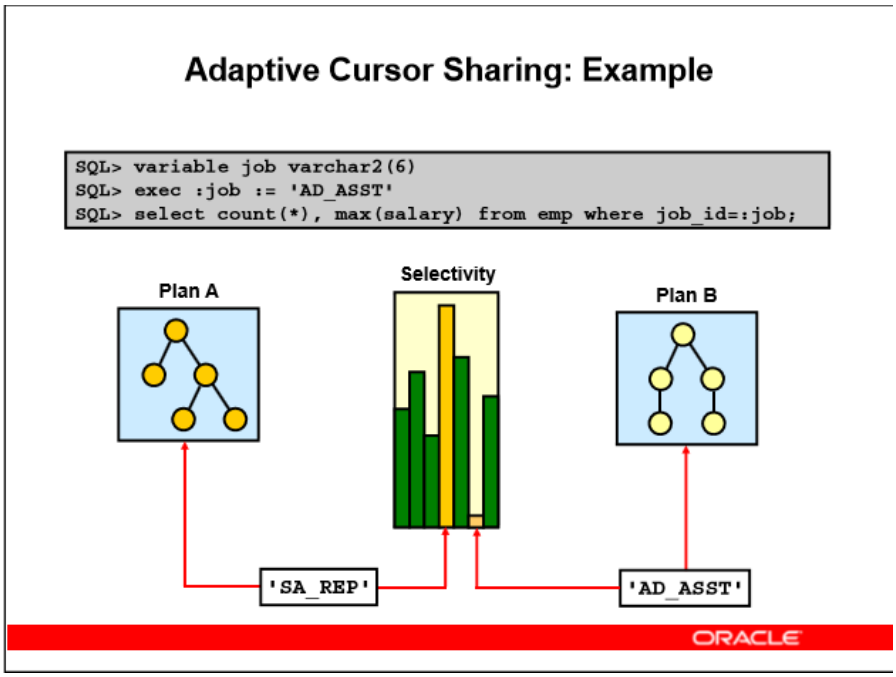
- IS_BIND_AWARE: Chỉ ra nếu một con trỏ được ràng buộc ý thức; giá trị YES | NO. Một con trỏ trong bộ nhớ cache con trỏ đã được chọn để sử dụng chia sẻ con trỏ ràng buộc ý thức được gọi là ràng buộc ý thức.

- V$SQL_CS_HISTOGRAM: Thể hiện các phân phối của việc thực hiện đếm trên toàn biểu đồ lịch sử thực hiện.

- V$SQL_CS_SELECTIVITY: Cho thấy mô hình lập phương chọn lọc được lưu trữ cho mỗi thuộc tính có chứa một biến ràng buộc và có chọn lọc được sử dụng trong kiểm tra con trỏ chia.

- V$SQL_CS_STATISTICS: Trình bày thống kê thực thi của một con trỏ bằng cách sử dụng bộ ràng buộc khác nhau.

7.2.4. Ví dụ



Xét dữ liệu trong hình, đây là biểu đồ thống kê trên cột JOB_ID, ta thấy rằng số lần xuất hiện của của SA_REP nhiều hơn rất nhiều AD_ASST. Trong trường hợp này, nếu các giá trị điều kiện được thay thế bởi một biến ràng buộc, câu truy vấn tối ưu sẽ thấy rằng số lượng giá trị của AD_ASST chiến chưa đầy 1 % trường giá trị trong bảng còn SA_REP chiến khoảng một phần ba. Nếu bảng có một triệu bản ghi thì với mỗi giá trị điều kiện khác nhau sẽ có 1 kế hoạch thực thi khác nhau.Các kết quả truy vấn AD_ASST là quét trong một miền giá trị bởi vì có rất ít hàng với giá trị đó. Các kết quả truy vấn SA_REP là quét toàn bảng vì rất nhiều trong số các hàng có giá trị, nó là hiệu quả hơn để đọc toàn bộ bảng. Nhưng, như đã có, việc sử dụng một biến ràng buộc gây ra các kế hoạch thực hiện tương tự được sử dụng cho cả hai giá trị, lần đầu tiên. Vì vậy, mặc dù có tồn tại kế hoạch khác nhau và tốt hơn cho mỗi người trong các giá trị, họ sử dụng cùng một kế hoạch.

Sau vài lần thực thi của câu truy vấn này khi sử dụng một biến ràng buộc, hệ thống xem xét truy vấn liên kết ý thức, lúc này nó thay đổi kế hoạch dựa trên các giá trị ràng buộc. Điều này có nghĩa là phương án tốt nhất được sử dụng cho các truy vấn, dựa trên các giá trị biến ràng buộc.

7.2.5. Tương tác với con trỏ thích nghi

Chia sẻ con trỏ thích nghi là độc lập với việc sử dụng tham số CURSOR_SHARING. Việc thiết lập các tham số này xác định xem các giá trị điều kiện được thay thế bởi các biến ràng buộc được hệ thống sinh ra. Nếu các tham số được thiết lập, chia sẻ con trỏ sẽ hoạt động theo tham số được thiết lập, nếu như người dùng cung cấp liên kết để bắt đầu.

Khi sử dụng các kế hoạch SPM (SQL Plan Management) nắm bắt tự động, kế hoạch đầu tiên được tạo trong một câu lệnh SQL với các biến ràng buộc được đánh dấu là kế hoạch SQL cơ bản tương ứng.Nếu kế hoạch khác được tìm thấy cho rằng câu lệnh SQL là tương tự nó được thêm vào câu lệnh SQL đã có kế hoạch thực thi và đánh dấu để xác thực. Nó sẽ không được sử dụng ngay lập tức. Vì vậy, mặc dù chia sẻ con trỏ thích nghi đã đưa ra một kế hoạch mới dựa trên một tập hợp mới các giá trị ràng buộc, SPM không cho phép nó được sử dụng cho đến khi kế hoạch đã được xác thực.Do đó quay trở lại hoạt động của Oracle10g, chỉ có kế hoạch được tạo ra dựa trên tập đầu tiên của các giá trị ràng buộc được sử dụng cho tất các câu lệnh tiếp theo. Một cách giải quyết có thể là chạy hệ thống cho một số kế hoạch nắm bắt tự động thiết lập để sai,và sau khi con trỏ cache đã được đưa tới tất cả các kế hoạch một câu lệnh SQL với các ràng buộc, tải toàn bộ kế hoạch trực tiếp từ con trỏ cache vào tương ứng kế hoạch SQL cơ sở.Bằng cách này, tất cả các kế hoạch cho một câu lệnh SQL đơn được đánh dấu như kế hoạch ban đầu SQL theo mặc định.

IIX. Tổng kết

Biến ràng buộc được đưa ra khái niệm bắt đầu từ oracle i8 và từ đó nó đã tạo ra một bước tiến lớn trong việc chia sẻ con trỏ. Biến ràng buộc và chia sẻ con trỏ giúp giảm được thời gian truy vấn, tiết kiệm được tài nguyên trong việc truy vấn, giúp tổng quát hóa được các câu truy vấn. Từ đó tạo ra nhiều lợi ích quan trọng trong truy vấn cơ sở dữ liệu.

Tuy vậy, sau những gì biến ràng buộc và con trỏ chia sẻ được trình bày ở trên, ta có thể rút ra một số điều:

- Chia sẻ con trỏ tiết kiệm bộ nhớ, giúp bộ tối ưu không phải làm việc nhiều với các câu truy vấn khác nhau giá trị điều kiện chọn, nhưng hiệu năng lại thực sự không tốt khi áp dụng một kế hoạch cho tất cả các câu truy vấn. Oracle11g đã đưa ra con trỏ thích nghi để giải quyết vấn đề này.
- Nếu sử dụng chia sẻ con trỏ thông thường, thì tùy vào việc phân bố dữ liệu của các giá trị điều kiện, ta mới quyết định có sử dụng biến ràng buộc hay không. Phân bố dữ liệu khác nhau có thể cần các kế hoạch thực hiện khác nhau để đạt được truy vấn tối ưu.
- Đôi khi ta cũng cần cân nhắc giữa hiệu quả và hiệu năng, ta chấp nhận 1 số câu truy vấn có hiệu năng thấp, bù lại nhận được hiệu quả từ việc sử dụng biến ràng buộc cho nhiều câu truy vấn khác. Rõ ràng lợi ích của biến ràng buộc và chia sẻ con trỏ là rất lớn, ta không thể vì một số câu bị hiệu năng thấp mà bỏ qua không sử dụng biến ràng buộc được. Nên suy xét giữa hiệu quả đem lại và ảnh hưởng tồi khi sử dụng biến ràng buộc.