*Tuning Indexes Performance*

Ví dụ về những câu SQL không tốt về performance:

1. SELECT \* FROM job\_history jh, employees e   
   WHERE substr(to\_char(e.employee\_id),2) = substr(to\_char(jh.employee\_id),2)
2. SELECT \* FROM employees  
   WHERE to\_char(salary) = :sal
3. SELECT \* FROM parts\_old  
   UNION  
   SELECT \* FROM parts\_new

* Trong ví dụ 1, câu query có hai lỗi: sử dụng nhiều tính toán phức tạp và không sử dụng được index
* Trong ví dụ 2, câu query có một lỗi: không sử dụng được index
* Trong ví dụ 3, câu query có hai lỗi: lấy toàn bộ dữ liệu và không sử dụng được index

Ngoài ra ở những câu trên có một lỗi chung là SELECT hết các field của bảng, điều này không cần thiết và sẽ gây lãng phí về tài nguyên cũng như performance giảm

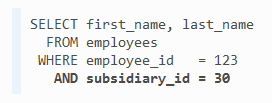
Những câu SQL chậm là những câu SQL ở phần điều kiện WHERE sử dụng:

1. Nhiều tính toán phức tạp
2. Convert ra thành các kiểu dữ liệu khác nhau cho mục đích so sánh
3. Không sử dụng được index trên chính cột được tạo index

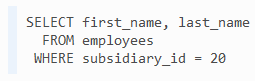
## Các tips để tạo index hiệu quả

1. **Một bảng nên có khóa chính**
   1. Giải thích: việc có khóa chính trong bảng sẽ ngầm báo cho DBMS đánh index lên cột này
   2. Chú ý: khi khai báo khóa chính là một tổ hợp gồm nhiều cột thì lúc này việc sắp xếp thứ tự các cột để đánh index cần được cân nhắc. [Xem mục 3](#Muc3)
2. **Phải có index trên khóa ngoại**
   1. Giải thích: việc có index trên khóa ngoại trong bảng sẽ giúp cho việc kết bảng nhanh hơn
3. **Thứ tự sắp xếp của các cột trong khóa chính ảnh hưởng đến việc index**: [xem ví dụ cụ thể](http://use-the-index-luke.com/sql/where-clause/the-equals-operator/concatenated-keys)
   1. Ngữ cảnh:



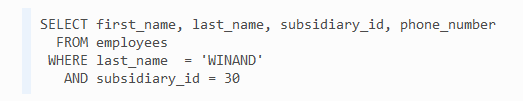
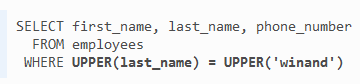
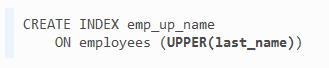
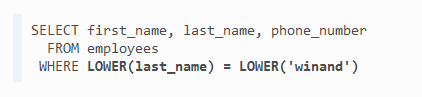
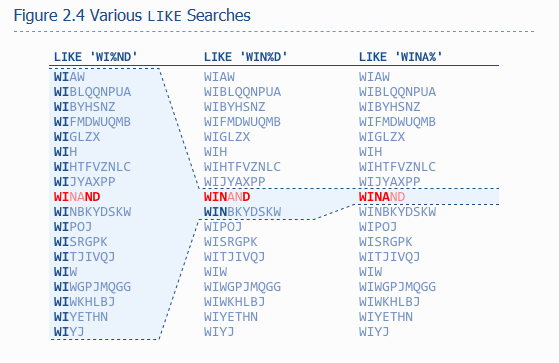


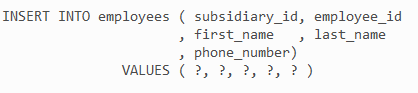
* 1. Vấn đề:



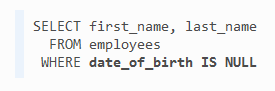
Câu query này không sử dụng được index vừa được tạo ở trên

* 1. Giải thích: khi chúng ta chỉ query mỗi subsidiary\_id thì việc đánh index theo thứ tự (employee\_id, subsidiary\_id) sẽ không phát huy tác dụng vì DBMS đang hiểu là sẽ search theo employee\_id sau đó mới tìm đến subsidiary\_id chứ không tìm subsidiary\_id. Vì vậy cách giải quyết cho vấn đề này là đảo lại thứ tự của việc tạo index thành (subsidiary\_id, employee\_id)
  2. Chú ý: chúng ta cũng có thể đánh index lên cột subsidiary\_id riêng nhưng khi đó bảng sẽ có nhiều index sẽ ảnh hưởng đến performance của việc delete, insert, update

1. **Trên bảng đã tạo index, cần chú ý nếu việc tạo index làm performance chậm hơn là quét toàn bảng**: [xem ví dụ cụ thể](http://use-the-index-luke.com/sql/where-clause/the-equals-operator/slow-indexes-part-ii)
   1. Ngữ cảnh:  
        
      
   2. Vấn đề: câu query lẽ ra không sử dụng index nhưng giờ đây lại dùng index và performance chậm hơn quét từng dòng
   3. Giải thích: Do cột last\_name không nằm trong index của bảng nên theo lý thuyết thì index sẽ không được sử dụng nhưng trong thực tế thì index ta mới tạo ra ở mục 3 lại được sử dụng và việc này lại chậm hơn việc chúng ta quét toàn bảng. Có hai vấn đề cần được hiểu ở đây
      1. Tại sao nó lại sử dụng index khi cột này không có trong danh sách index
         1. Giải thích: do bộ explainer lấy thông tin cấu hình không phù hợp nên xảy ra việc dùng index trên câu query này.
      2. Tại sao việc xài index lại chậm
         1. Giải thích: do DBMS sau khi tìm đc dòng data thỏa subsidiary\_id thì sẽ bắt đầu quét từng dòng dựa vào rowID có sẵn của DB kết hợp với việc load lượng lớn dữ liệu để quét sẽ làm cho việc sử dụng index chậm hơn với quét toàn bộ bảng
2. **Sử dụng hàm built-in trong mệnh đề WHERE sẽ không sử dụng được index**: [xem ví dụ cụ thể](http://use-the-index-luke.com/sql/where-clause/functions/case-insensitive-search)
   1. Ngữ cảnh:  
      
   2. Vấn đề: Không sử dụng được index trên cột last\_name
   3. Giải thích: Lúc này tên cột trở thành tham số của hàm nên sẽ không còn mối liên hệ giữa tên cột và giá trị tìm kiếm mà thay vào đó chúng ta cần phải đánh index lên mối quan hệ giữa hàm và giá trị tìm kiếm nghĩa là tạo index lên UPPER(last\_name). Tuy nhiên việc tạo ra index như vầy là không hợp lý vì vậy ta có thể tránh việc này bằng cách lúc tạo bảng ta sẽ cho cột này là cột được tính toán trước
   4. Chú ý:
      1. Một vài ORM tự thêm vào hàm UPPER hoặc LOWER mà người dùng không biết. Điền hình là Hibernate sẽ tự thêm vào LOWER cho trường hợp case-insensitive
      2. Ta cần phải update lại statistic cho bộ optimizer.
3. **Không nên sử dụng nhiều index lên một bảng**: [xem ví dụ cụ thể](http://use-the-index-luke.com/sql/where-clause/functions/over-indexing)
   1. Ngữ cảnh:   
      
   2. Vấn đề:   
        
      Câu query này không sử dụng được câu index phía trên
   3. Giải thích: nếu như ta có thêm một câu query dùng hàm LOWER thì việc ta đánh index như ở mục 5 là vô nghĩa. Vì vậy kinh nghiệm rút ra ở đây là chúng ta cần phải nhất quán khi sử dụng trong điều kiện so sánh
   4. Chú ý: chúng ta chỉ nên đánh index lên cột của bảng chứ không nên đánh index cho hàm
4. **Cân nhắc sử dụng LIKE với WILDCARD để câu query được hiệu quả**: [xem ví dụ cụ thể](http://use-the-index-luke.com/sql/where-clause/searching-for-ranges/like-performance-tuning)
   1. Giải thích: Mệnh đề LIKE sẽ kết hợp với WILDCARD để tìm kiếm, việc để WILDCARD càng ở cuối từ được query sẽ giúp cho tốc độ truy vấn tăng nhanh vì số lượng record được load lên sẽ giảm đáng kể  
      
   2. Chú ý: với trường hợp WILDCARD ở đầu keyword tìm kiếm, ta có thể việc không sử dụng được index bằng cách dùng hàm contains được xây dựng sẵn của Oracle, khi này ta cần phải chỉ định kiểu index của cột cần query là context. [Xem thêm ví dụ chi tiết](http://www.dba-oracle.com/t_sql_contains_clause.htm)
5. **Để sử dụng index lên cột NULL, ta cần thêm vào một cột không thể NULL trong index**: [xem ví dụ cụ thể](http://use-the-index-luke.com/sql/where-clause/null/index)
   1. Ngữ cảnh: bảng employees cho phép cột date\_of\_birth có giá trị NULL  
      

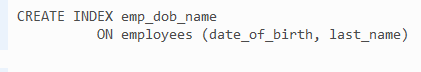


Insert các dòng dữ liệu ko có dữ liệu về date\_of\_birth

* 1. Vấn đề: Khi dùng câu query này thì index không sử dụng được  
     
  2. Giải thích: Oracle không hỗ trợ index cho những record có giá trị NULL. Mặt khác, do cách tạo index là (date\_of\_birth) nên khi cần query với điều kiện date\_of\_birth IS NULL thì DBMS sẽ chỉ quét được những record có giá trị date\_of\_birth, còn những record ko có thì DBMS sẽ không sử dụng index này ([xem lại mục 3](#Muc3)). Vì vậy để giải quyết vấn đề này ta có thể chỉ định thêm một cột luôn không bao giờ NULL để tạo index.  
     Cách 1:



Cách 2:



1. **Việc sử dụng ràng buộc NOT NULL có thể ảnh hưởng đến performance**. [Xem ví dụ cụ thể](http://use-the-index-luke.com/sql/where-clause/null/not-null-constraint)
   1. Giải thích: Khi ta dùng một cột (cột last\_name) không bao giờ NULL để index cho cột có khi bị NULL (cột date\_of\_birth) như trường hợp số 8 được đề cập đến ở trên thì khi ta bỏ ràng buộc NOT NULL trên last\_name thì sẽ làm cho index của date\_of\_birth ko hoạt động. Vì vậy khi bỏ đi ràng buộc ta cần cân nhắc những ảnh hưởng lên các index đang sử dụng cột đó.
   2. Chú ý: đối với những câu query xài dùng hàm để so sánh trong mệnh đề WHERE thì ta nên chọn giải pháp tạo giá trị đó ngay từ lúc insert để khi query ta không cần phải qua giai đoạn tính toán để so sánh giá trị