Tối ưu hóa index

Tối ưu hóa index

- Index và Join

Chiến lược Join – Ví dụ

- Quan hệ: R và S
  - kích thước 1 block: 4kB
  - R: $n_r = 5000$ records, $b_r = 100$ disk blocks, $0.4MB$
  - S: $n_s = 10000$ records, $b_s = 400$ disk blocks, $1.6MB$
- Chạy ví dụ: $R \bowtie S$
  - R được gọi là quan hệ ngoài
  - S được gọi là quan hệ trong

Chiến lược Join – Naïve Nested Loop

- Naïve nested loop join
  - Lấy mỗi bản ghi của R (quan hệ ngoài) và tìm qua tất cả các bản ghi thích hợp của S (quan hệ trong)
  - Đối với mỗi bản ghi của R, S được duyệt
- VD: naïve nested loop join
  - Trường hợp xấu nhất: buffer có thể giữ một block của mỗi quan hệ
  - R được duyệt mỗi lần, S được duyệt n_r lần
  - Tổng cộng $n_r b_s + b_r$ = 2,000,100 blocks phải được đọc (=8GB)
  - Chú ý: trường hợp xấu nhất khác nếu S là quan hệ ngoài.
  - Trường hợp tốt nhất: cả 2 quan hệ khớp với bộ nhớ chính
  - $b_r + b_s$ = 500 block cần đọc

Chiến lược Join – Block Nested Loop

- Block nested loop join
  - So sánh tất cả các hàng của mỗi block của R đến tất cả bản ghi trong S
  - Đối với mỗi block của R, S được duyệt.
- VD:
  - Trường hợp xấu nhất: buffer có thể giữ chỉ một block cho mỗi quan hệ
  - R được quét một lần, S được quét b_r lần
  - Tổng cộng: $b_r b_s + b_r$ = 40,100 block cần được đọc (=160MB)
  - Trường hợp tốt nhất: $b_s + b_r$ = 500 block được đọc

Chiến lược Join – Index Nested Loop

- Index nested loop join
  - Lấy mỗi hàng của R và quan sát tương ứng trong S sử dụng index
  - Runtime là O(|R| x log|S|) (vs. O(|R| x |S| của naïve nested loop)
  - Hiệu quả nếu index cover join (không dữ liệu truy xuất trong S)
  - Hiệu quả nếu R có ít bản ghi hơn S có trang nhớ: không phải tất cả trang nhớ của S được đọc (vd: foreign key join từ bảng nhỏ tới lớn
- VD:
  - B+-tree index trên S có 4 lớp, nên max c=5 truy xuất trên một bản ghi của S.
  - Tổng cộng: $b_r + n_r c = 25{,}100$ block cần được đọc (=100MB)

Chiến lược Join – Merge Join

- Merge join (hai nhóm index)
  - Duyệt R và S trong thứ tự sắp xếp order và merge.
  - Mỗi block của R và S được đọc một lần
- Không index trên R và/hoặc S
  - Nếu không index: sắp xếp và lưu trữ quan hệ với $b(2\lceil log_{M-1}(b/M)\rceil + 1) + b$ block dịch chuyển (M: free memory block)
  - Nếu non-cluster index hiện tại: có thể index duyệt
- VD:
  - Trường hợp tốt nhất: nhóm index trên R và S (M=2 là đủ)
  - $b_r + b_s$ = 500 block cần được đọc (2MB)
  - Trường hợp xấu nhất: không index, chỉ M = 3 memory block
  - Sắp xếp và lưu trữ R (1400 block) và S (7200 block) trước: join với 9100 936MB) block chuyển tổng cộng
  - Trường hợp: M = 25memory block: 2500 block chuyển (10MB)

Chiến lược Join – Hash Join

- Hash join (tương đương, không index)
  - băm cả 2 bảng đến các nhóm sử dụng cùng một hàm băm
  - join theo cặp của các nhóm tương ứng trong bộ nhớ chính
  - R được gọi là probe input, S được gọi là build input
- Joining bucket trong bộ nhớ chính:
  - build hash index trên một nhóm từ S (với hàm băm mới)
  - probe hash index với tất cả bộ dữ liệu trên nhóm tương ứng của R
  - nhóm build phải khớp với bộ nhớ chính, nhóm probe không cần
- VD:
  - giả sử mỗi nhóm probe khớp trong bộ nhớ chính
  - R và S được duyệt để tính toán nhóm, các nhóm được ghi vào bộ nhớ sau đó được đọc theo cặp
  - Tổng coongj3(br + bs) = 1500 block được đọc/ghi (6MB)
  - Mặc định trong SQLServer và DB2 UDB khi không index đại diện

Distinct Values và Join Selectivity

- Join Selectivity
  - Số cặp lấy ra chia bởi tính chất của sản phẩm chéo
  - Selectivity là thấp nếu kết quả join nhỏ
- Giá trị distinct dẫn đến các thuộc tính join trên một bảng $(|R \bowtie S|/|R \times S|)$
- Hiệu suất giảm với số giá trị distinct join
  - Một vài giá trị distinct trong cả 2 bảng thường nghĩ là nhiều bản ghi giống nhau
  - Nhiều bản ghi giống nhau: kết quả join lớn, join chậm
  - Hash join: nhóm lớn (nhóm build không khớp với bộ nhớ chính)
  - Index join: các bản ghi giống nhau trên multiple trang nhớ
  - Merge join: các bản ghi giống nhau không khớp trong bộ nhớ ở cùng thời điểm

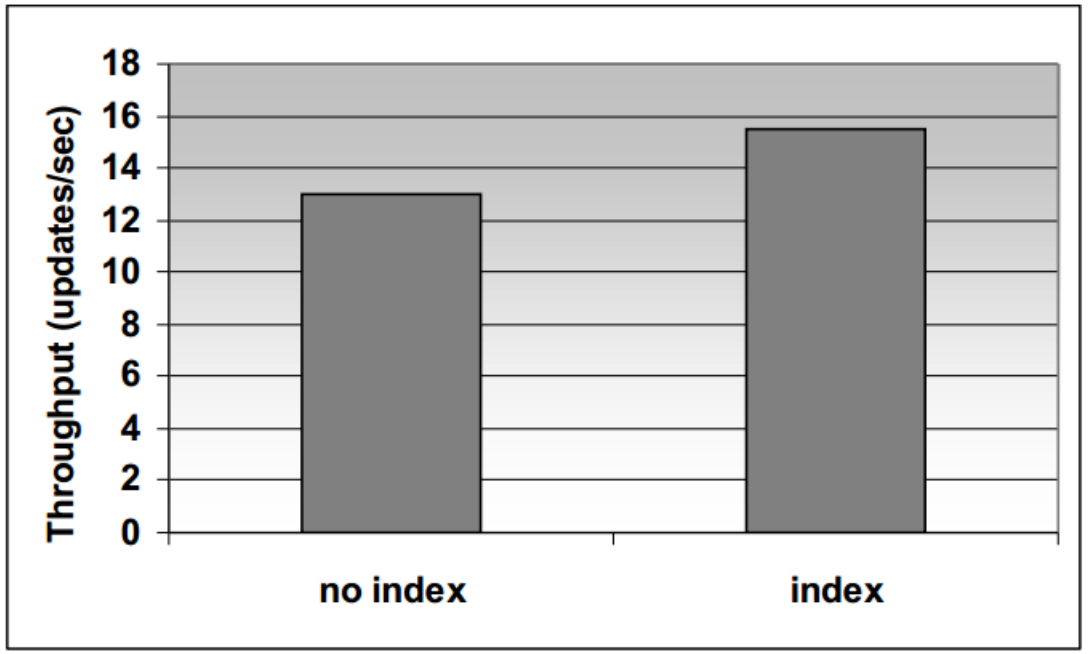
Foreign Keys

- Foreign key: thuộc tính R.A lưu trữ key của bảng khác, S.B
- Hạn chế của foreign key: R.A phải được thiết lập giống S.B
  - insert trong R phải kiểm tra xem foreign key có tồn tại trong S
  - delete trong S phải kiểm tra có bản ghi nào có key đó trong R không
- Index giúp kiểm tra foreign key dễ dàng hơn
  - index trên R.A tăng tốc delete từ S
  - index trên S.B tăng tốc insert vào R
  - một số hệ thống có thể tạo index trên R.A và/hoặc S.B mặc định
- Foreign key join:
  - mỗi bản ghi của một bảng giống nhiều nhất một bản ghi của bảng khác
  - join gần như thường xuyên trong thực nghiệm
  - cả hash và index nested loop đều hiệu quả

Index on Small Table

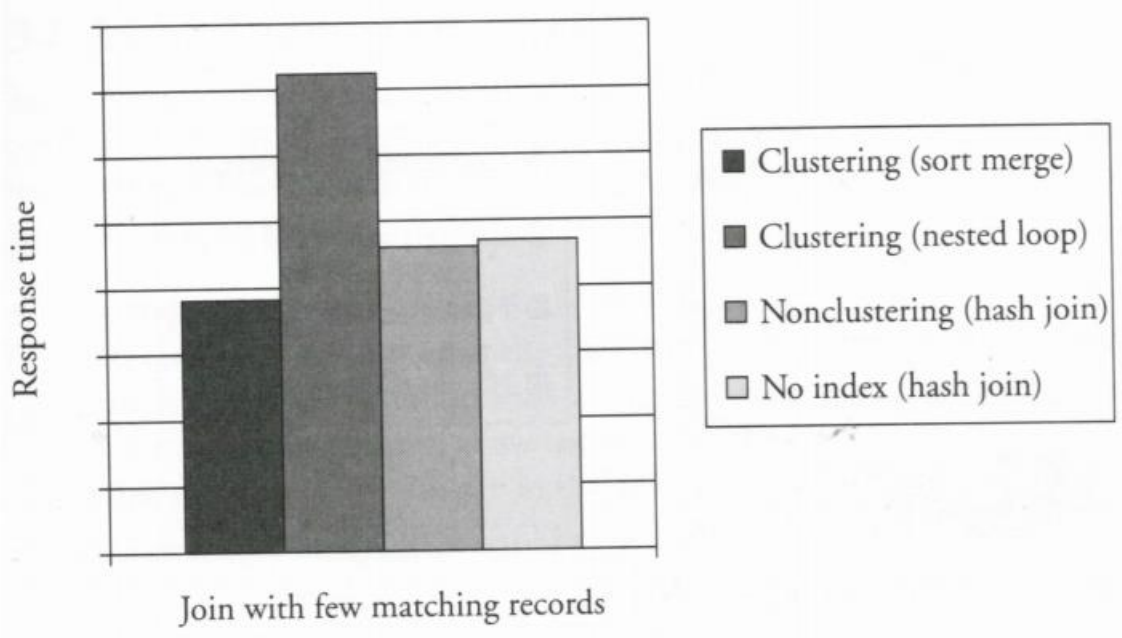
- Đọc truy vấn trên các bản ghi nhỏ:
  - các bảng có thể khớp trên một track đơn trên bộ nhớ
  - đọc truy vấn yêu cầu một lần tìm
  - index không hữu dụng: tìm ít nhất một trang index và một trang table
- Table với bản ghi lớn (~ kích thước trang)
  - mỗi bản ghi chiếm gần cả trang
  - VD: 200 bản ghi chiếm 200 trang
  - index hữu dụng cho point query (đọc 3 trang so với 200)
- Nhiều insert và delete:
  - index phải được tái tổ chức (khóa)
  - khóa xung đột gần root từ khi index là nhỏ
- Cập nhật của các bản ghi đơn
  - không cần index, table phải được duyệt
  - các bản ghi đã duyệt bị khóa
  - duyệt (và do đó tranh chấp khóa) có thể tránh với index

Update Queries on a Small Table



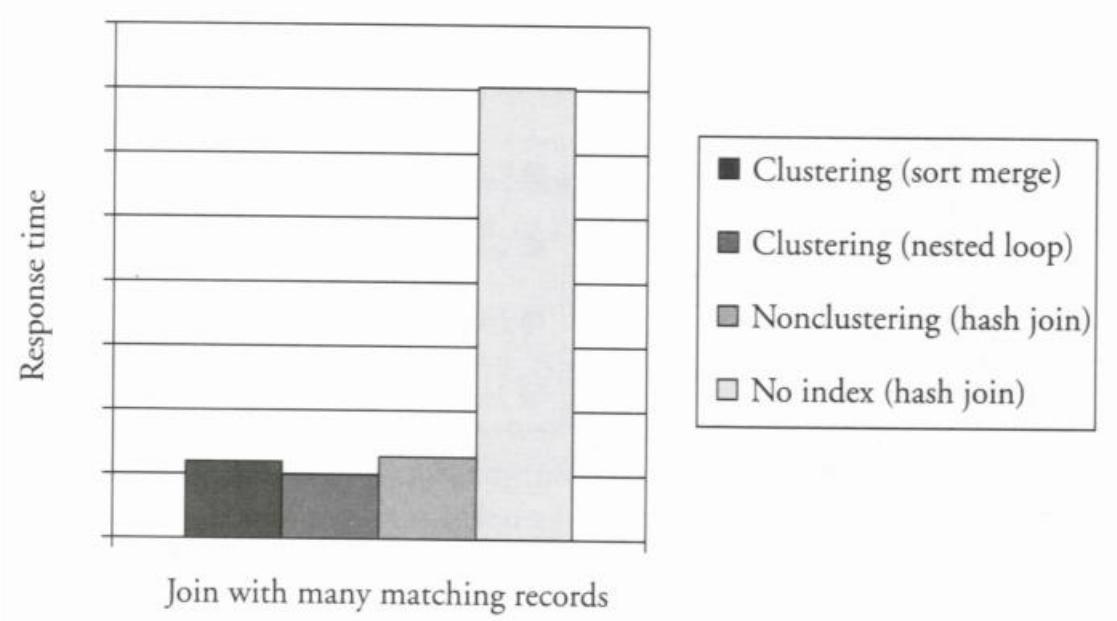
Index tránh duyệt bảng và tranh chấp khóa

Thử nghiệm - Join với vài bản ghi giống nhau



non-clustered index được bỏ qua, hash join được sử dụng thay thế

Thử nghiệm – Join với nhiều bản ghi giống nhau



- ✓ Tất cả join chậm từ khi kích thước output là lớn
- ✓ Hash join (no index) chậm bởi vì nhóm là rất lớn