Tối ưu hóa index

Các loại truy vấn

- Các index khác nhau phù hợp với các loại truy vấn khác nhau.
- Point query: trả về nhiều nhất 1 bản ghi

```
SELECT name
FROM Employee
WHERE ID = 8478
```

- Multipoint query: trả về nhiều bản ghi dựa trên điều kiện tương ứng

```
SELECT name
FROM Employee
WHERE department = 'IT'
```

- Range query: trả về các bản ghi với giá trị trong khoảng thời gian có giá trị.

```
SELECT name
FROM Employee
WHERE salary >= 155000
```

- Kết hợp truy vấn tiền tố: cho một chuỗi các lệnh thuộc tính, truy vấn xác định một điều kiện về một tiền tố của một chuỗi thuộc tính
- VD: chuỗi thuộc tính: lastname, firstname, city

  - Các câu truy vấn sau là truy vấn kết hợp tiền tố:

    - `lastname='Gates'`
    - `lastname='Gates' AND firstname='George'`
    - `lastname='Gates' AND firstname like 'Ge%'`
    - `lastname='Gates' AND firstname='George' AND city='San Diego'`

  - Các câu truy vấn sau không phải là truy vấn kết hợp tiền tố:

    - `firstname='George'`
    - `lastname LIKE '%ates'`

- Truy vấn cực trị: trả về các bản ghi với max hoặc min các giá trị của một số thuộc tính.

```
SELECT name
FROM Employee
WHERE salary = MAX(SELECT salary FROM Employee)
```

- Truy vấn đặt: yêu cầu các bản ghi từ các giá trị thuộc tính.

```
SELECT *
FROM Employee
ORDER BY salary
```

- Truy vấn nhóm: nhóm các bản ghi lại với nhau; thông thường mỗi hàm sẽ được áp dụng trên mỗi nhóm.

```
SELECT dept, AVG(salary)
FROM Employee
GROUP BY dept
```

➢ Truy vấn kết hợp: kết hợp 2 hoặc nhiều hơn 1 bảng.

- Join cân bằng:

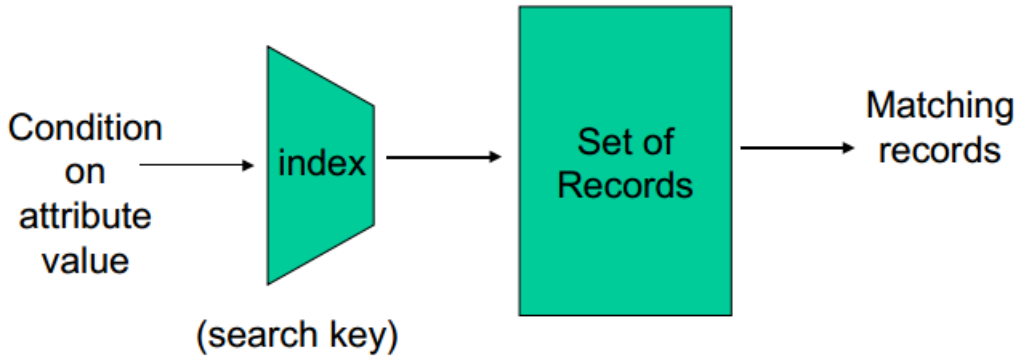
```
SELECT Employee.ssnum
FROM Employee, Student
WHERE Employee.ssnum = Student.ssnum
```

- Join với các điều kiện không cân bằng:

```
SELECT e1.ssnum
FROM Employee e1, Employee e2
WHERE e1.manager = e2.ssnum
AND e1.salary > e2.salary
```

Thế nào là một index?

- Một index là một cấu trúc dữ liệu mà hỗ trợ hiệu quả để truy nhập đến dữ liệu:



- Tối ưu hóa index rất cần thiết cho hiệu suất.
- Lựa chọn index không đúng cách có thể dẫn đến:
  - Các index mà được chọn có thể không được sử dụng
  - Nhiều file được quét để trả về chỉ 1 bản ghi>
  - Nhiều bảng cùng join có thể khiến chương trình chạy nhiều giờ.

Key của một index

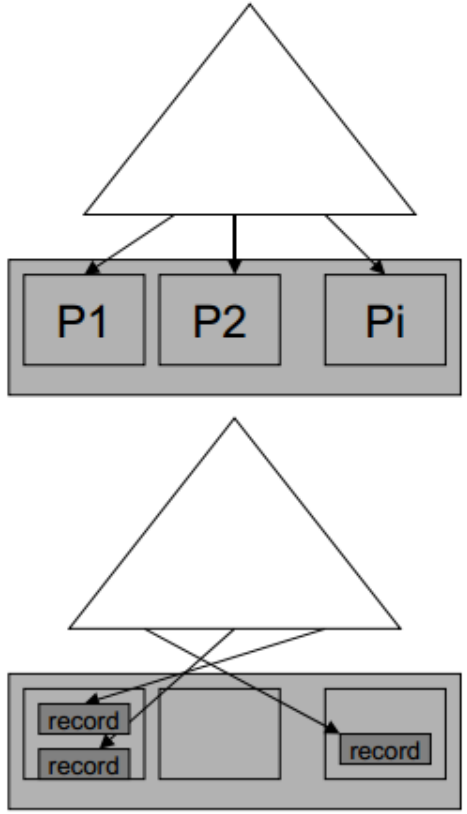
- Search key: hoặc key đơn giản của một index là:
  - Đơn thuộc tính hoặc nhóm thuộc tính.
  - Các giá trị của thuộc tính khóa chính dùng để truy xuất vào bản ghi trong bảng.
- Sequential key:
  - Giá trị đơn trong các lệnh INSERT: time stamp, counter,…
- Non-sequential key:
  - Giá trị không liên quan đến lệnh INSERT: social security number, last name,…
- Chú ý: index key khác key trong lý thuyết chuẩn
  - Key thuộc tính: có giá trị duy nhất
  - Index key: không cần thiết phải là duy nhất.

Đặc điểm của index

- Các index thường được nhìn như các cây(B+ tree, hash)
  - Một vài node trong main memory. (root,…)
  - Một số node thấp hơn của cây ít có khả năng trong main memory.
- Số mức: là số của node tính từ node gốc trong đường đi từ gốc đến nó.
  - Một node thường là một block trong bộ nhớ
  - Một block đọc bắt buộc cho mỗi mức.
  - Đọc một block chiếm vài phần nghìn giây (bao gồm cả tìm vùng nhớ)
- Fanout: số node con mà 1 node có thể có.
  - Fanout lớn tức là mức càng thấp (gần root)
- Chiến lược tràn: insert into a full node n
  - B+ - tree: chia n thành n và n’, cả 2 đều có khoảng cách đến root giống nhau.
  - Tràn chuỗi: n lưu con trỏ tới node mới n’

Sparse and Dense

- Sparse index: các con trỏ tới các trang bộ nhớ
  - Ít nhất một con trỏ trên một trang
  - Thường ít con trỏ hơn là các bản ghi
- Dense index: các con trỏ đến các bản ghi riêng biệt
  - Một key trên một bản ghi
  - Thường nhiều key hơn là sparse index
  - Tối ưu hóa: lưu các key trùng nhau chỉ một lần theo các con trỏ.



Sparse and Dense

- STT của con trỏ:
  ptrs in dense index = records per page * ptrs in sparse index
- Pro sparse: ít pointers
  - Thông thường kích thước bản ghi nhỏ hơn kích thước trang
  - Ít con trỏ dẫn đến mức thấp hơn (và truy xuất bộ nhớ)
  - Sử dụng ít không gian hơn
- Pro dense: index có thể "bao gồm" truy vấn

Covering index

- Covering index:
  - Trả lời các truy vấn đọc trong cấu trúc index
  - Nhanh, tính từ lức dữ liệu chưa được truy xuất
- VD1: dense index on lastname

  SELECT COUNT(lastname) WHERE lastname=‘Smith’
- VD2: dense index on A, B, C (theo thứ tự này)

- covered query:

```
SELECT B, C
FROM R
WHERE A = 5
```

- covered query, but not prefix:

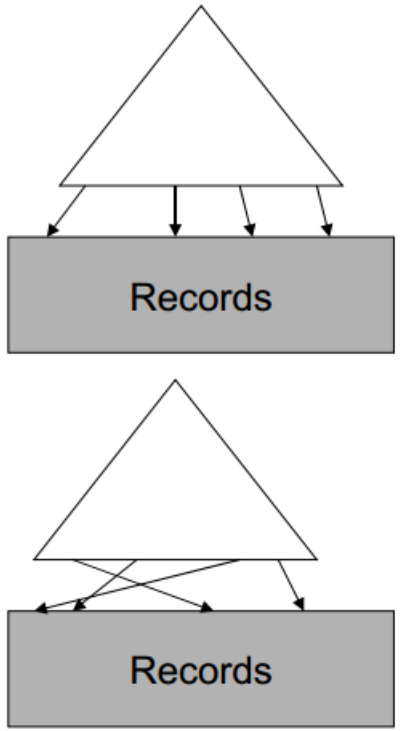
```
SELECT A, C
FROM R
WHERE B = 5
```

- non-covered query: D requires data access

```
SELECT B, D
FROM R
WHERE A = 5
```

Clustering vs. Non-Clustering

- Clustering index trên thuộc tính X (primary index)
  - Các bản ghi được nhóm bằng thuộc tính X trong bộ nhớ
  - B+-tree: các bản ghi được sắp xếp bởi thuộc tính X.
  - Chỉ một clustering index trên một bảng.
  - Dense hoặc sparse.
- Non-clustering index trên thuộc tính X (secondary index)
  - Không có rang buộc về tổ chức bảng.
  - Có nhiều hơn một index trên một bảng
  - Luôn luôn là dense.



Clustering index

- Có thể được sparse:
  - Ít con trỏ hơn non-clustering index.
  - Nếu bản ghi có kích thước nhỏ, lưu một truy xuất bộ nhớ trên một truy xuất bản ghi
- Tốt cho multi-point query:
  - Truy xuất ngang bằng trên các thuộc tính không duy nhất.
  - Tất cả bản ghi kết quả trên các trang nhớ liên tiếp.
  - VDL hãy xem lastname trên phonebook.
- Tốt cho range, prefic, ordering query:
  - Hoạt động nếu clustering index được khai báo như B+-tree.
  - Tiền VD: xem xét tất cả lastname bắt đầu với ‘St’ trong danh bạ điện thoại
  - Bản ghi trả về nằm trên các trang nhớ liên tiếp.
- Tốt cho equality join:
  - Nhanh ngay cả với thuộc tính không phải key.
  - Index trên một bảng: là index trong vòng lặp lồng
  - Index trên cả hai bảng: merge-join
- overflow pages làm giảm hiệu quả:
  - Nếu trang bộ nhớ đã đày, sẽ dẫn đến lỗi overflow pages.
  - overflow pages khi mà có thêm quá nhiều truy xuất bộ nhớ.

Equality Join với Clustering Index

VD:

```
SELECT Employee.ssnum, Student.course
FROM Employee, Student
WHERE Employee.firstname = Student.firstname
```

- Index trên Employee.firstname: sử dụng index của join vòng lặp lồng
  - Với mỗi student, tìm các employee với firstname giống nhau
  - Tất cả các employee thỏa mãn trên các trang liên tiếp
- Index trên tất cả thuộc tính firstnam: sử dụng merge join
  - Đọc tất cả các bảng trong danh sách sắp xếp (B+-tree)
  - Mỗi trang được đọc một lần
  - Hoạt động với cả hash index cùng với hàm băm tương ứng.

Clustering Index và Overflow Pages

- Tại sao lại overflow pages?
  - Clustering index lưu các bản ghi trên các trang nhớ liên tiếp.
  - Insert giữa hai trang nhớ liên tiếp là không thể
  - Nếu trang nhớ đầy, tràn bản ghi dẫn đến overflow pages (tràn trang)
- Các truy xuất bộ nhớ khi overflow page: làm giảm tốc độ.
- Overflow pages có thể là do:
  - Insert
  - Update mà có thay đổi giá trị key
  - Update mà tăng kích thước bản ghi (vd: thay thế NULL bằng string)
- Tổ chức lại index:
  - Gọi các công cụ đặc biệt
  - Hoặc đơn giản là hủy và tạo lại index.

Chiến lược tràn

- Tối ưu vùng nhớ trống trên trang nhớ:
  - Oracle, DB2: pctfree(0 là đầy), SQLServer: fillfactor(100 là đầy)
  - Vùng nhớ trống trên trang được sử dụng cho những bản ghi mới.
  - Vùng nhớ trống nhỏ: không gian nhớ đang được tận dụng hiệu quả, tốc độ đọc sẽ nhanh hơn
  - Còn quá nhiều vùng nhớ trống: giảm nguy cơ tràn
- Chiến lược tràn:
  - Split: chia một trang thành 2 nửa trang và link trang mới
    VD: A→B→C, chia thành A→B'→B''→C (SQLServer)
  - Chaining: trang đầy trỏ tới trang tràn (Oracle)
  - Append: bản ghi tràn của tất cả các trang được nối thêm vào cuối của bảng (DB2)

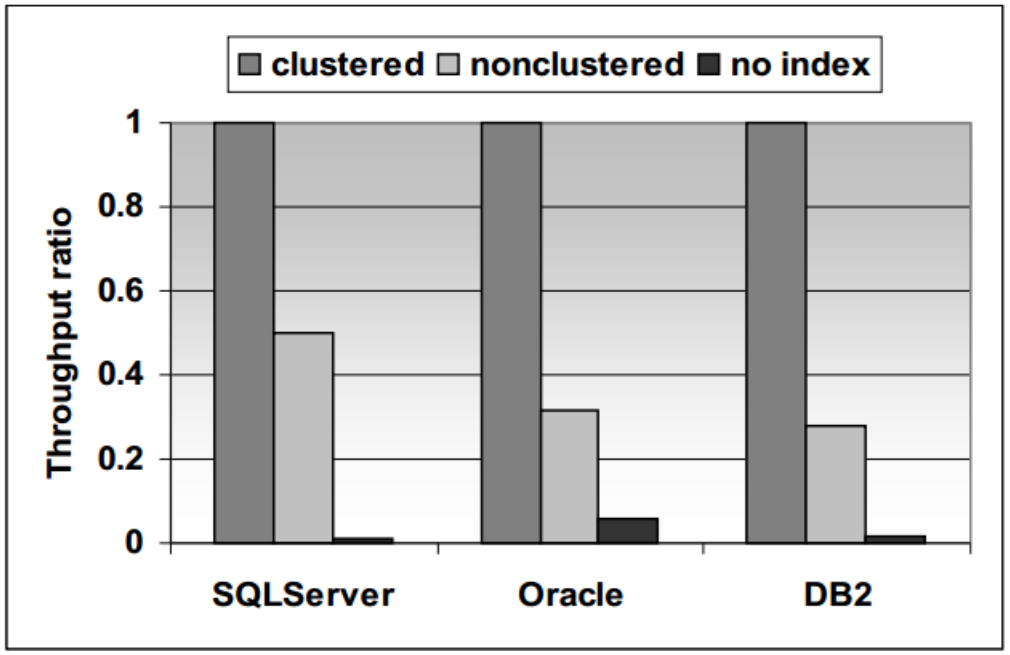
Non-clustering Index

- Luôn hữu dụng cho truy vấn điểm.
- Hữu dụng nếu index che lấp truy vấn.
- Critical table: bao gồm index trên tất cả các thuộc tính kết hợp có liên quan.
- Multi-point query (not covered): chỉ khi không quá nhiều SELECT
  - nR: số bản ghi trả về bởi truy vấn
  - nP: số trang bộ nhớ trong bảng.
  - Số bản ghi nR được phân bố đều trên các trang.
  - Nên truy vấn sẽ đọc min(nR, nP) trang bộ nhớ.
- Index có thể làm chậm: nhiều SELECT trong truy vấn multi-point
  - Quét là phép nhanh hơn là truy xuất vào tất cả các trabg với index.
  - Nên nR nhỏ hơn đáng kể so với nP.

VD của non-lustering index và truy vấn multi-point.

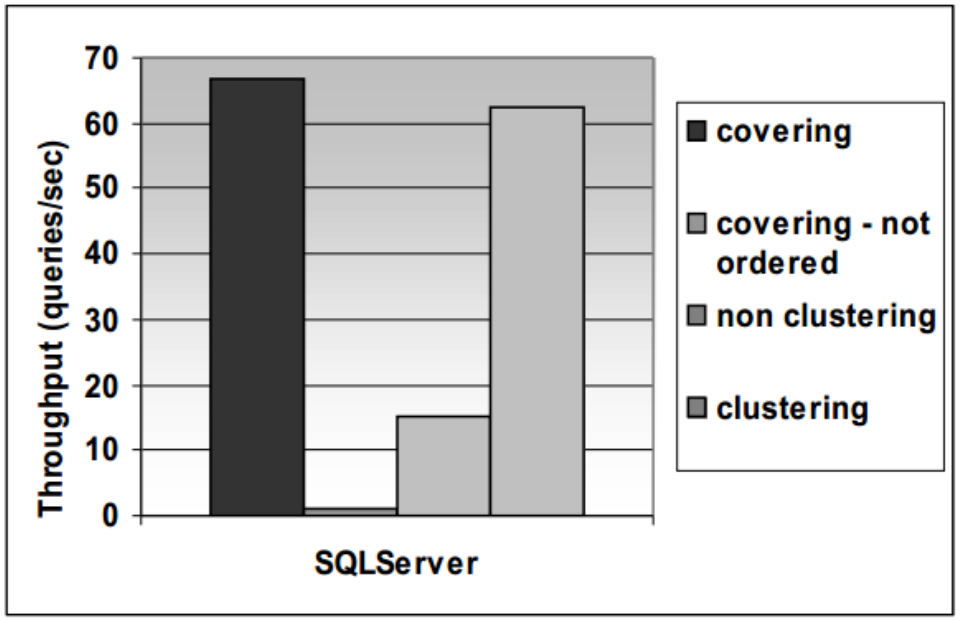
- Example 1:
  - records size: $50B$
  - page size: $4kB$
  - attribute A takes 20 different values (evenly distributed among records)
  - does non-clustering index on A help?
- Evaluation:
  - $nR = m/20$ (m is the total number of records)
  - $nP = m/80$ (80 records per page)
  - $m/20 > m/80$ thus index does not help
- Example 2: as above, but record size is $2kB$
- Evaluation:
  - $nR = m/20$ (m is the total number of records)
  - $nP = m/2$ (2 records per page)
  - $m/20 << m/2$ thus index might be useful

Clustering vs. non-clustering index



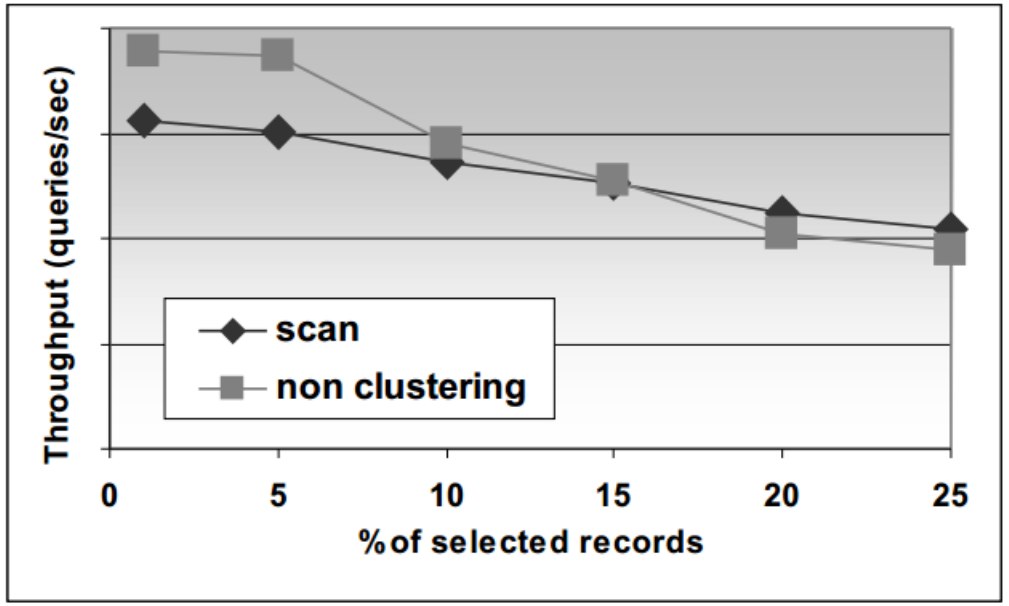
- ✓ Multi –point query với selectivity 100/1M bản ghi (0.01%)
- ✓ Clustering index nhanh hơn rất nhiều non-clustering index
- ✓ Full table scan (no index) các order của cường độ chậm hơn index.

Covering vs. Non-covering index



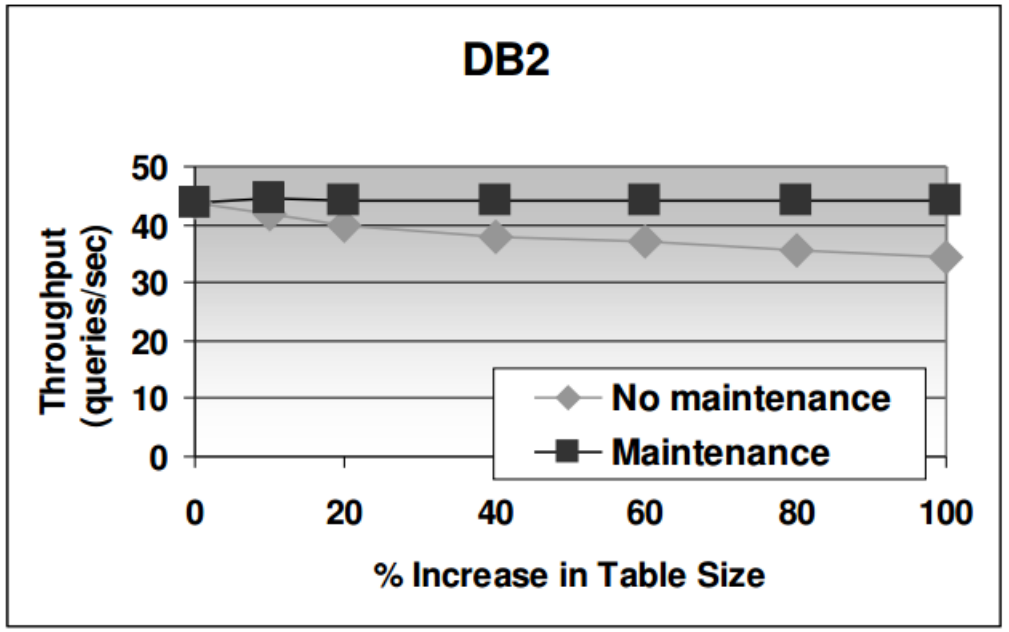
- ✓ Tiền tố kết hợp truy vấn vào chuỗi các thuộc tính
- ✓ Covering: index cover truy vấn, điều kiện truy vấn trên tiền tố
- ✓ Covering, not order: index cover query, nhưng điều kiện không tiền tố
- ✓ Non-clustering: non-covering index, điều kiện truy vấn trên tiền tố
- ✓ Clustering: sparse index, điều kiện truy vấn trên tiền tố

Non-covering vs. Table Scan



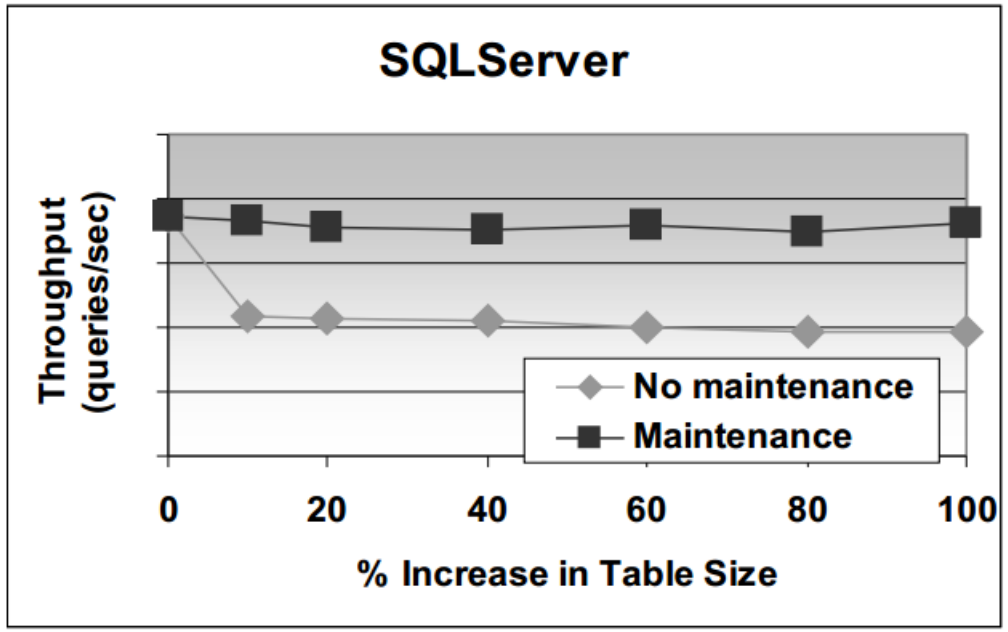
- ✓Query: range query
- ✓Non clustering: non-clustering non-covering index
- ✓Scan: no index, vd: table scan required
- ✓Index nhanh hơn nếu ít hơn 15% của các bản ghi được chọn

Index Maintenance – DB2



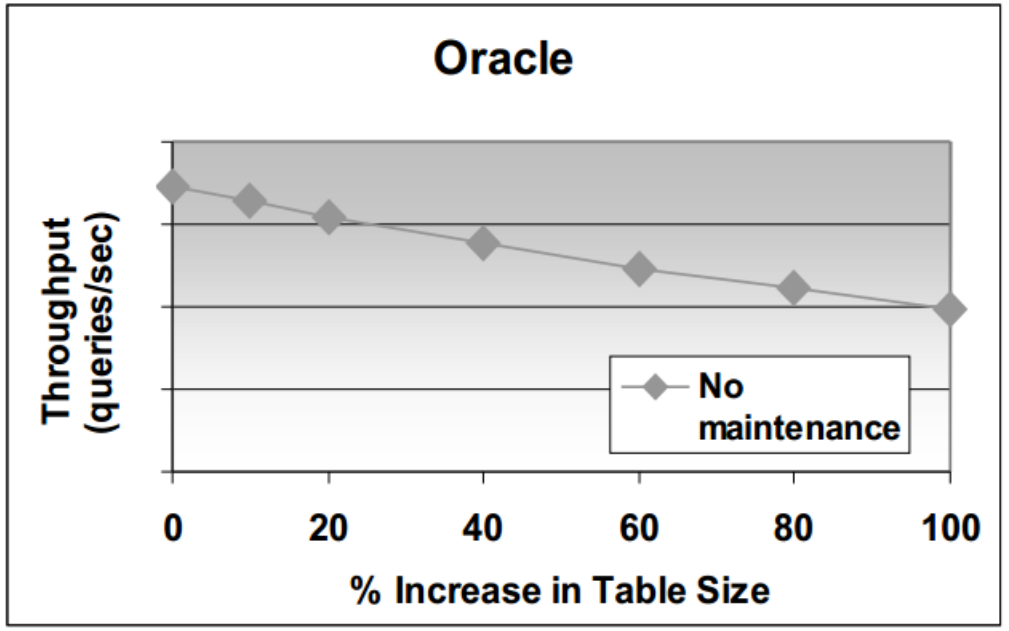
- ✓ Query: hàng loạt 100 multi-point query, pctfree=0 (data page đầy)
- ✓ Làm giảm hiệu suất với INSERT
- ✓ Overflow records simply appended
- ✓ Truy vấn đi qua index và sau đó quét tất cả các bản ghi overflow
- ✓ Reorganization helps.

Index Maintenance – SQL Server



- ✓ Fillfactor = 100 (data page đầy)
- ✓ Hiệu suất giảm với INSERT
- ✓ Chuỗi tràn được duy trì cho trang tràn
- ✓ Mở rộng truy xuất bộ nhớ
- ✓ Reorganization helps.

Index Maintenance – Oracle



- ✓ Pctfree = 0 (data page đầy), performance giảm với INSERT
- ✓ Tất cả index trong Oracle là non-clustering
- ✓ Index-organized table là clustered bởi primary key
- ✓ Recreateing index không tổ chức lại bảng
- ✓ Maintenance: xuất và tái nhập bảng để tái tổ chức.