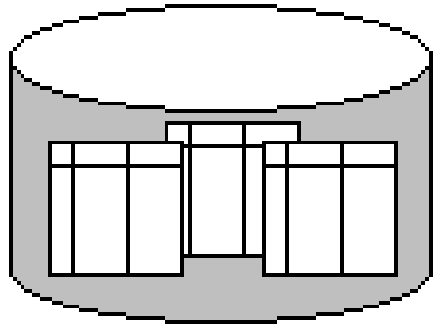
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Συστήματα Διαχείρισης Βάσεων Δεδομένων

Φροντιστήριο 6: Index Tuning

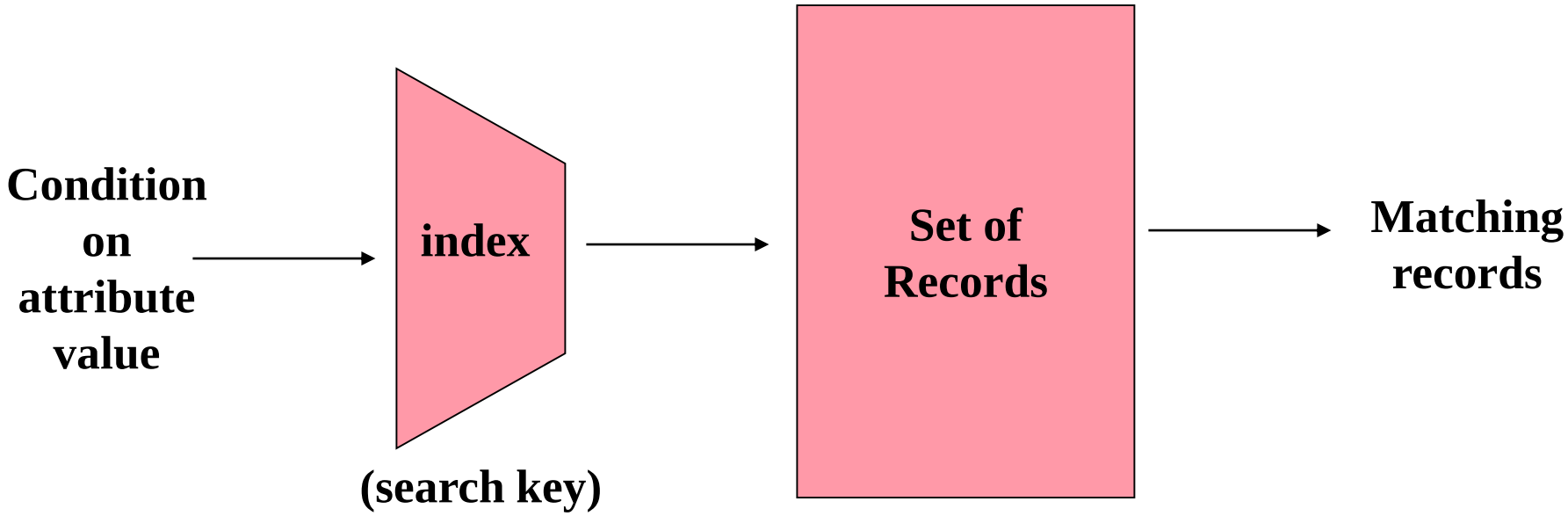
Δημήτρης Πλεξουσάκης
Τμήμα Επιστήμης Υπολογιστών

INDEX TUNING



Index

An index is a data structure that supports efficient access to data



- Different indexes are good for different query types.
- We identify categories of queries with different index requirements.

Index Performance Issues

- Type of Query
- Index Data Structure
- Organization of data on disk
- Index Overhead
- Data Distribution
- Covering

Types of Queries

❶ Point Query : returns at most one record based on equality condition

```
SELECT balance
FROM accounts
WHERE number = 1023;
```

❷ Multipoint Query: returns multiple records based on equality condition

```
SELECT balance
FROM accounts
WHERE branchnum = 100;
```

❸ Range Query: on X returns records with values in interval of X

```
SELECT number
FROM accounts
WHERE balance > 10000;
```

❹ Prefix Match Query: given an attribute or a sequence of attributes X, specifies only a prefix of X.

```
SELECT *
FROM employees
WHERE  name = 'Jensen'
       and firstname LIKE 'Ca%'
       and age < 30;
```

Types of Queries

❺ Extremal Query: returns records with max or min values on some attributes

```
SELECT *
FROM accounts
WHERE balance =
  max(select balance from accounts)
```

❻ Ordering Query: orders records by some attribute value

```
SELECT *
FROM accounts
ORDER BY balance;
```

❼ Grouping Query: partition records into groups; usually a function is applied on each partition

```
SELECT branchnum, avg(balance)
FROM accounts
GROUP BY branchnum;
```

❽ Join Query: link two or more tables

```
SELECT distinct branch.adresse
FROM accounts, branch
WHERE
  accounts.branchnum =
     branch.number
and accounts.balance > 10000;
```

Search Keys

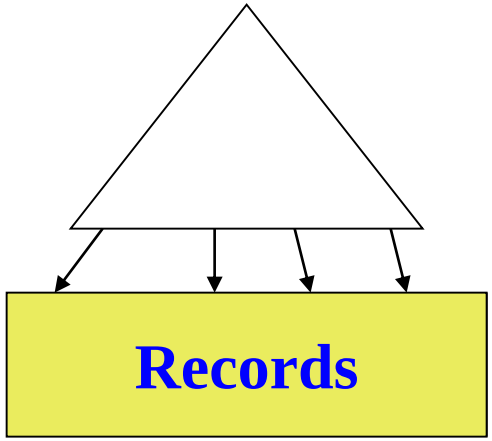
- A (search) key is a sequence of attributes.
- Types of keys
  - Sequential: the value of the key is monotonic with the insertion order (e.g., counter or timestamp)
  - Non sequential: the value of the key is unrelated to the insertion order (e.g., social security number)

Indexes

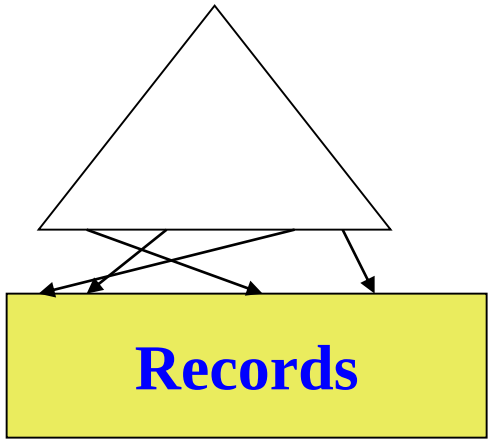
- B+-trees
  - B+-Tree is a balanced tree whose leaves contain a sequence of key-pointer pairs.
- Hash table (hash map)
  - Hash is a method of storing key-value pairs based on a pseudo-randomizing function called a hash function.
- R-trees
  - R-tree is an index used for spatial access methods, i.e., for indexing multi-dimensional information (geography or geometry).
- Bitmaps
  - Bitmap is simply an array of bits.
- T-trees
  - A T-tree is a balanced index tree data structure optimized for cases where both the index and the actual data are fully kept in memory.

Clustered – Non clustered index

- Clustered index (primary index)
  - A clustered index on attribute X co-locates records whose X values are near to one another.
  - There might be only one clustered indexes per table.

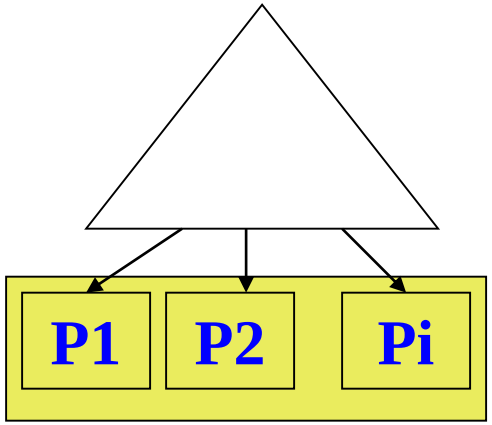


- Non-clustered index (secondary index)
  - A non clustered index does not constrain table organization.
  - There might be several non-clustered indexes per table.

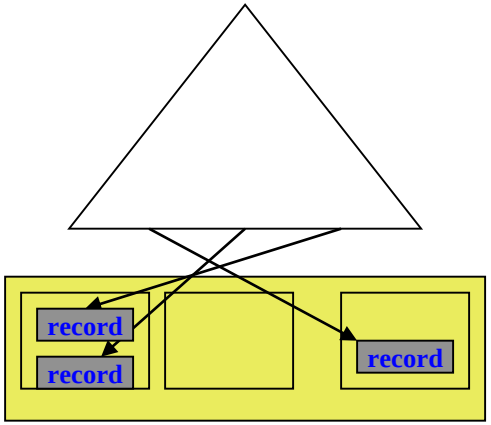


Sparse/Dense Index

- Sparse index
  - Pointers are associated to pages



- Dense index
  - Pointers are associated to records
  - Non clustered indexes are dense



Index Implementations in some major DBMS

- SQL Server
  - B+-Tree data structure
  - Clustered indexes are sparse
  - Indexes maintained as updates/insertions/deletes are performed
- DB2
  - B+-Tree data structure, spatial extender for R-tree
  - Clustered indexes are dense
  - Explicit command for index reorganization
- Oracle
  - B+-tree, hash, bitmap, spatial extender for R-Tree
  - Clustered index
    - Index organized table (unique/clustered)
    - Clusters used when creating tables.
- TimesTen (Main-memory DBMS)
  - T-tree

Index Tuning -- data

- Settings:

```
employees(ssnum, name, lat, long, hundreds1, hundreds2);
  clustered index c on employees(hundreds1)
   with fillfactor = 100;
  nonclustered index nc on employees (hundreds2);
  index nc3 on employees (ssnum, name, hundreds2);
  index nc4 on employees (lat, ssnum, name);
```

  - 1000000 rows ;
  - Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000

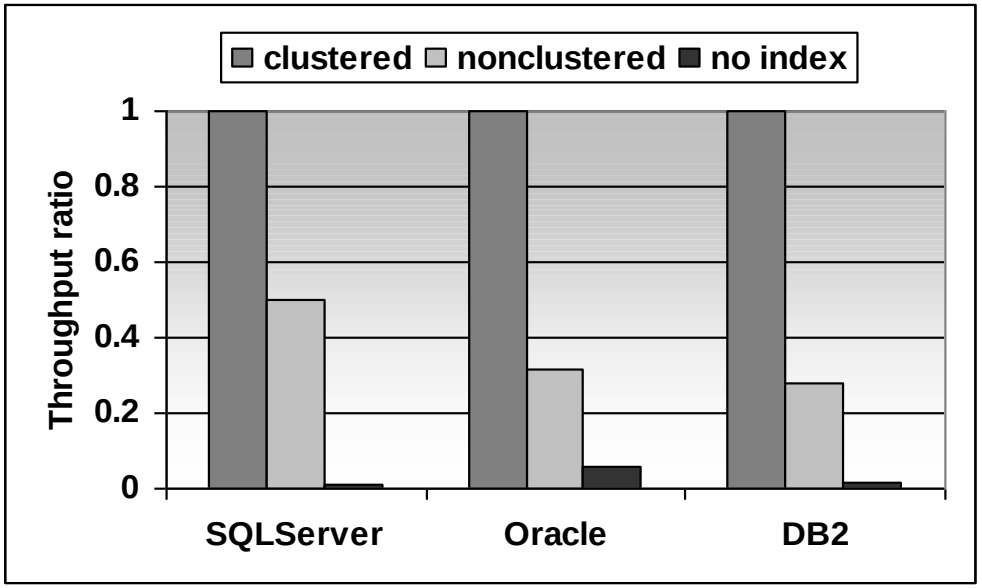
Index Tuning -- operations

- Operations:
  - Update:
    ```
    update employees set name = 'XXX' where ssnum = ?;
    ```
  - Insert:
    ```
    insert into employees values
    (1003505,'polo94064',97.48,84.03,4700.55,3987.2);
    ```
  - Multipoint query:
    ```
    select * from employees where hundreds1= ?;
    select * from employees where hundreds2= ?;
    ```
  - Range Query:
    ```
    select * from employees where long between ? and ?;
    ```
  - Point Query:
    ```
    select * from employees where ssnum = ?
    ```

Clustered Index

- Benefits of a clustered index:
  - A sparse clustered index stores fewer pointers than a dense index
    - This might save up to one level in the B-tree index
  - A clustered index is good for multipoint queries
    - White pages in a paper telephone book
  - A clustered index based on a B-Tree supports range, prefix, extremal and ordering queries well
  - A clustered index (on attribute X) can reduce lock contention:
    - Retrieval of records or update operations using an equality, a prefix match or a range condition based on X will access and lock only a few consecutive pages of data
- Cost of a clustered index
  - Due to insertions
  - Cost of overflow pages
  - Due to updates (e.g., a NULL value by a long string)

Clustered Index



- Multipoint query that returns 100 records out of 1000000
- Clustered index is twice as fast as non-clustered index and orders of magnitude faster than a scan

Positive Points of Clustering indexes

- If the index is sparse, it has less points --less I/Os
  - Good for multipoint queries
    - e.g. Looking up names in telephone dir
  - Good for equijoin. Why?
  - Good for range, prefix match, and ordering queries

- Because there is only one clustered index per table, it might be a good idea to replicate a table in order to use a clustered index on two different attributes
  - Yellow and white pages in a paper telephone book
  - Low insertion/update rate

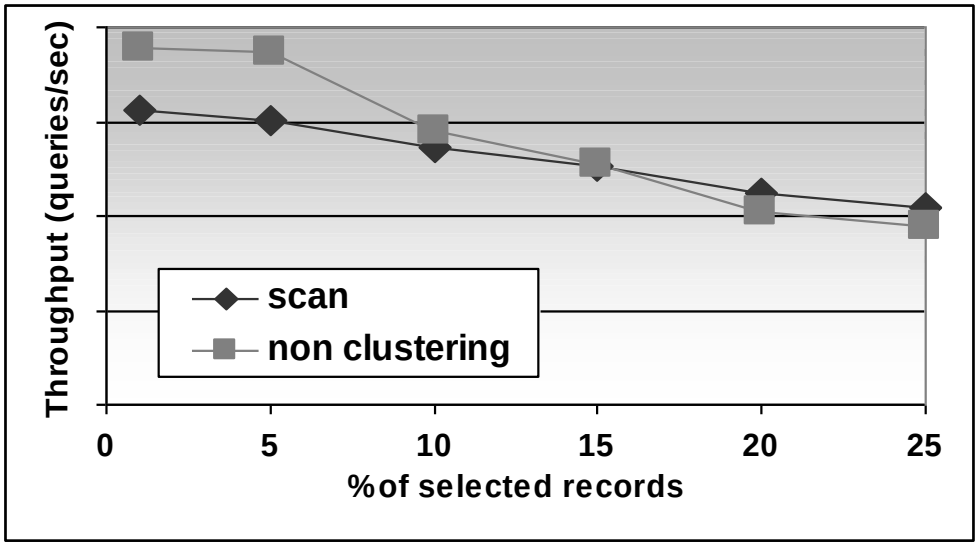
Non-Clustered Index

- Benefits of non-clustered indexes
  - A dense index can eliminate the need to access the underlying table through covering
    - It might be worth creating several indexes to increase the likelihood that the optimizer can find a covering index
- A non-clustered index is good if each query retrieves significantly fewer records than there are pages in the table
  - Point queries
  - Multipoint queries:
    - number of distinct key index values > c * number of records per page
    - Where *c* is the number of pages that can be prefetched in one disk read

Positive/negative points of non-clustering indexes

- Eliminate the need to access the underlying table
  - eg. Index on (A, B, C)
  - `select B,C from R where A=5`
- Good if each query retrieves significantly fewer records than there are pages in DB
- May not be good for multipoint queries

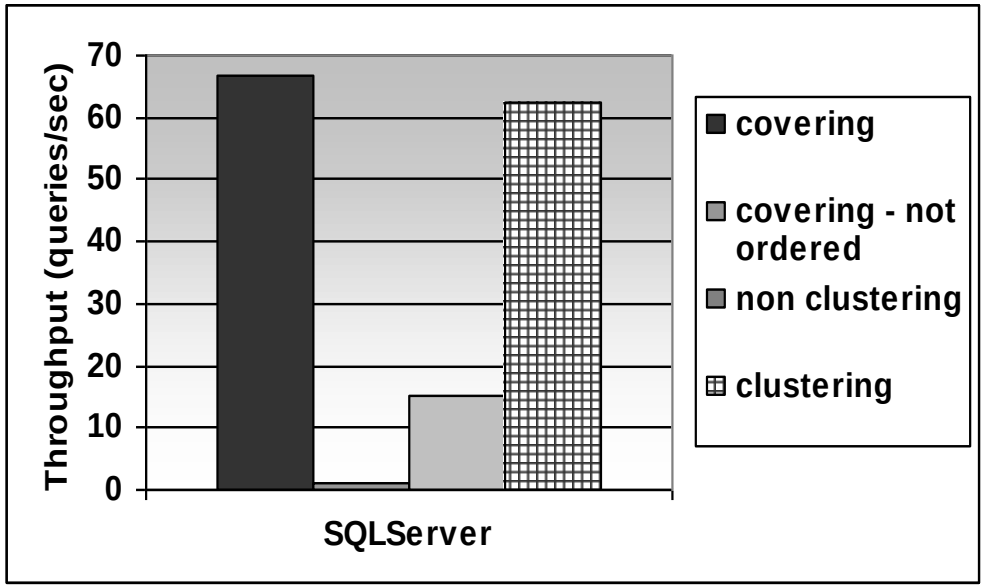
Scan Can Sometimes Win



- IBM DB2 v7.1 on Windows 2000
- Range Query
- If a query retrieves 10% of the records, scanning is often better than using a non-clustering non-covering index.
  Crossover > 10% when records are large or table is fragmented on disk – scan cost increases.

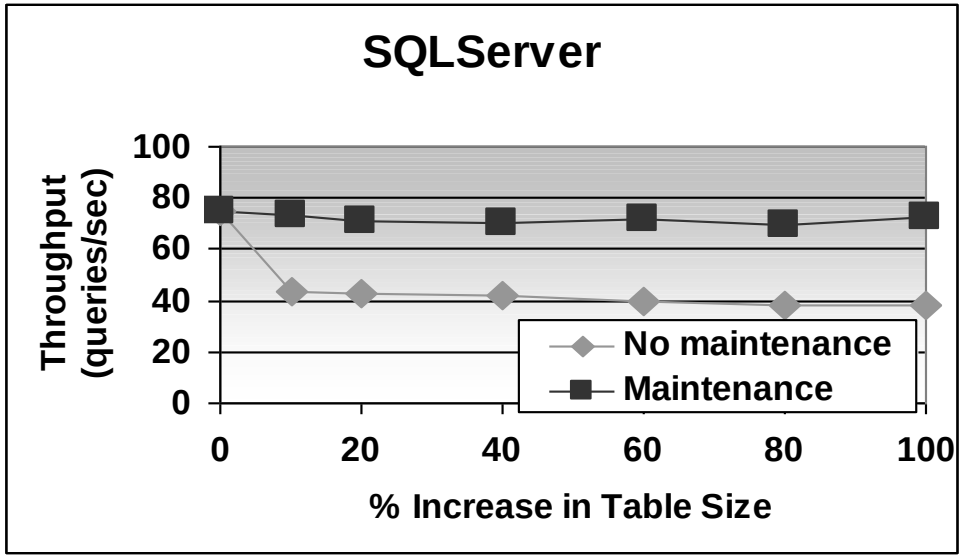
Covering Index

- `Select name from employee where department="marketing"`
  - Good covering index would be on (department, name)
  - Index on (name, department) less useful
  - Index on department alone moderately useful



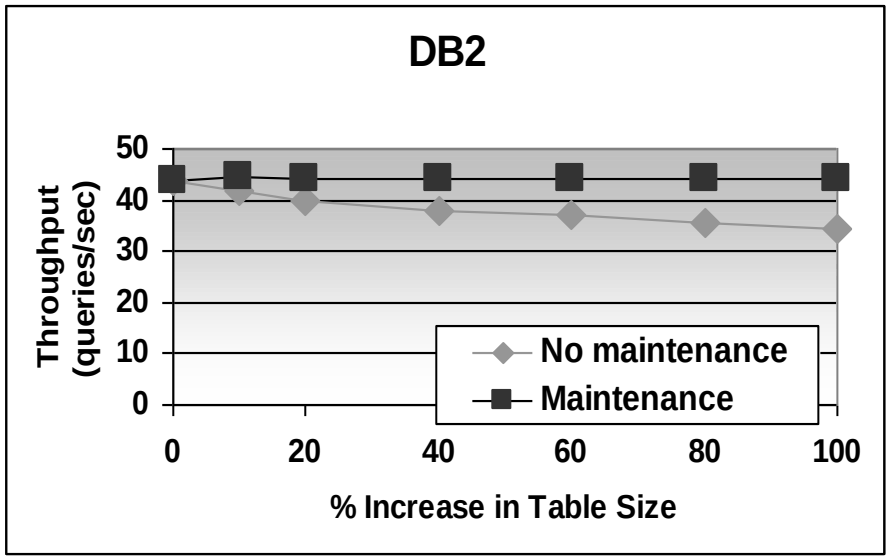
- Covering index performs better than clustering index when first attributes of index are in the where clause and last attributes in the select
- When attributes are not in order then performance is much worse

Evaluation of clustered indexes with insertions



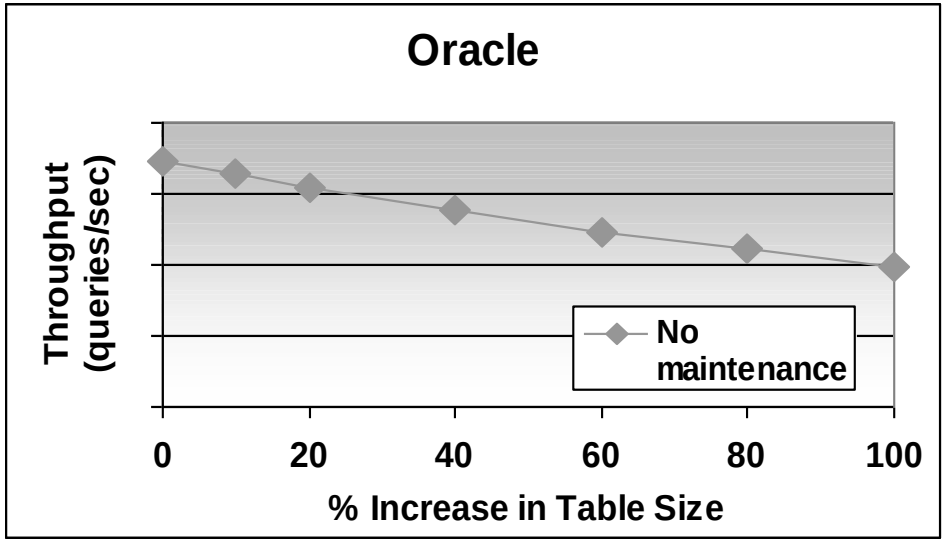
- Index is created with fillfactor= 100
- Insertions cause page splits and extra I/O for each query
- Maintenance consists in dropping and recreating the index
- With maintenance performance is constant while performance degrades significantly if no maintenance is performed

Evaluation of clustered indexes with insertions



- Index is created with pctfree = 0
- Insertions cause records to be appended at the end of the table
- Each query thus traverses the index structure and scans the tail of the table
- Performances degrade slowly when no maintenance is performed

Evaluation of clustered indexes with insertions



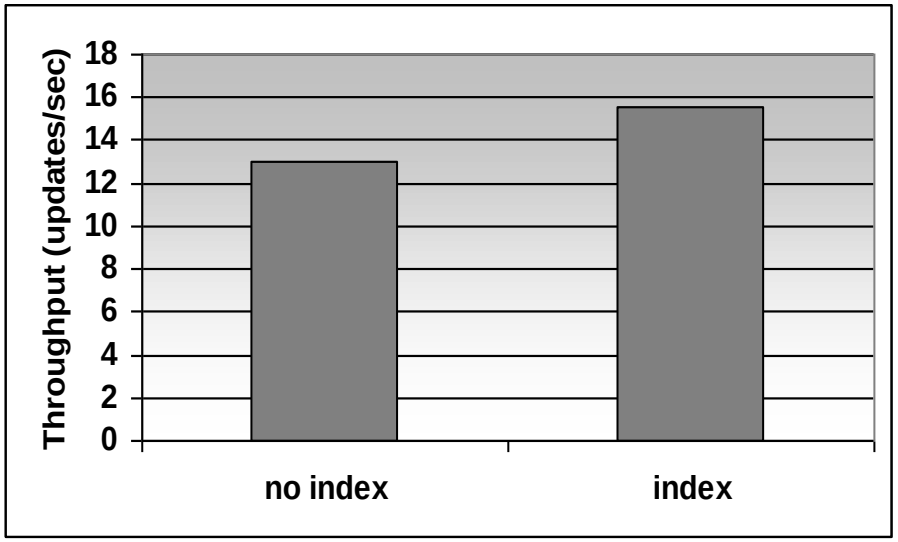
- In Oracle, all indexes are non-clustered, clustered index are approximated by an index defined on a clustered table
- No automatic physical reorganization (maintenance)
- Index defined with pctfree = 0
- Overflow pages cause performance degradation

Index on Small Tables

- Tuning manuals suggest to avoid indexes on small tables
  - If all data from a relation fits in one page then an index page adds an I/O
  - If each record fits in a page then an index helps performance

Index on Small Tables

If transactions update a single record, without an index, each transaction scans through many records before it locks the relevant record, thus reducing update concurrency



- Small table: 100 records, i.e., a few pages
- Two concurrent processes perform updates (each process works for 10ms before it commits)
- No index: the table is scanned for each update. No concurrent updates
- A clustered index allows to take advantage of row locking

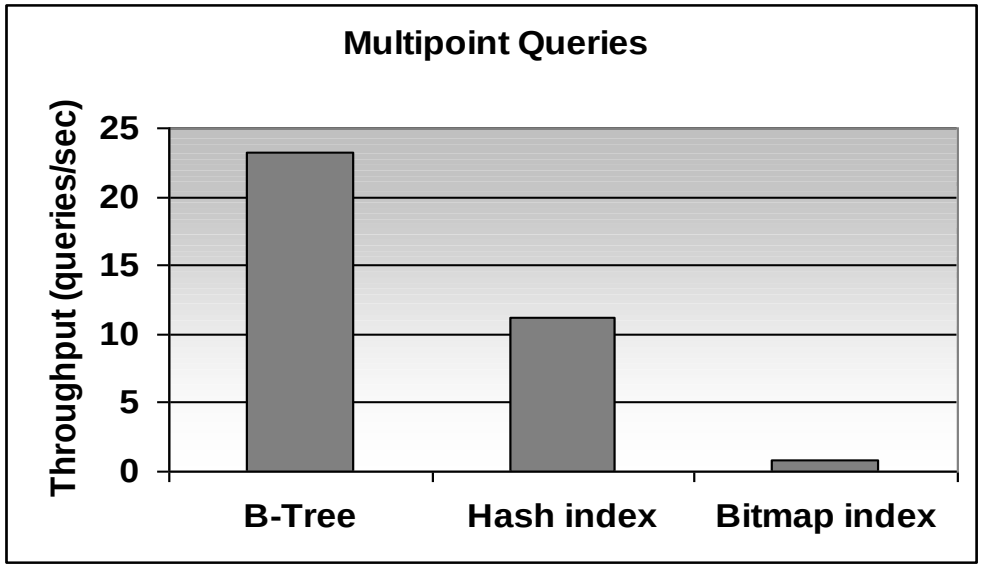
Bitmap vs. Hash vs. B+-Tree

- Settings:

```
employees(ssnum, name, lat, long, hundreds1, hundreds2);
  create cluster c_hundreds (hundreds2 number(8))
    PCTFREE 0;
  create cluster c_ssnum(ssnum integer) PCTFREE 0 size
    60;
  create cluster c_hundreds(hundreds2 number(8))
    PCTFREE 0 HASHKEYS 1000 size 600;
  create cluster c_ssnum(ssnum integer) PCTFREE 0
    HASHKEYS 1000000 SIZE 60;
  create bitmap index b on employees (hundreds2);
  create bitmap index b2 on employees (ssnum);
```

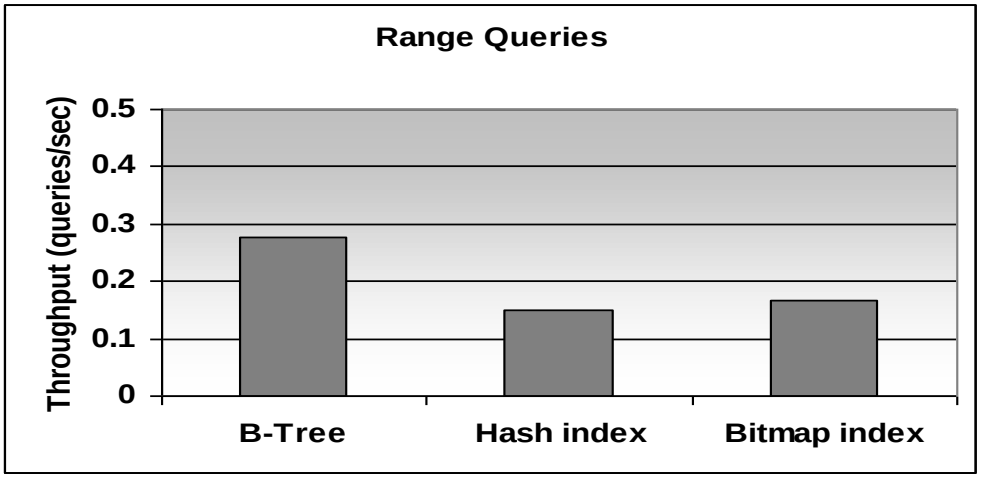
  - 1000000 rows ;
  - Dual Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000

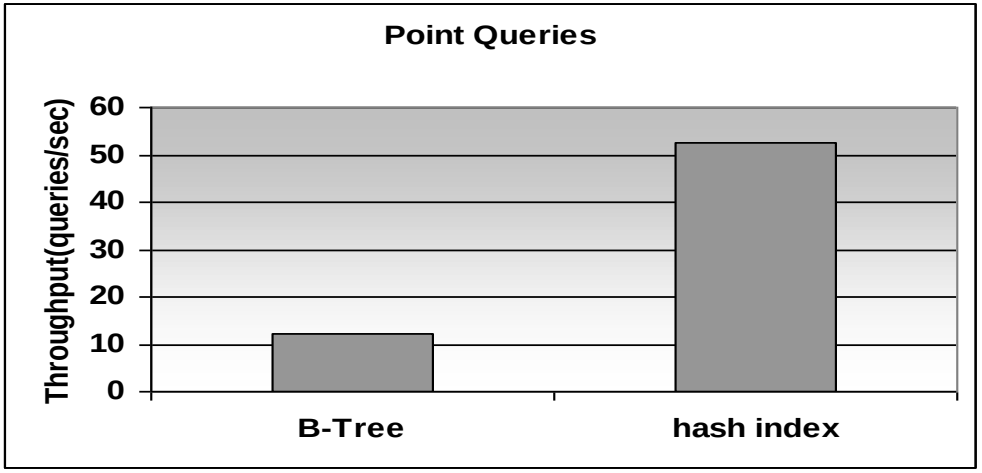
Multipoint Query: B-Tree, Hash Tree, Bitmap



- There is an overflow chain in a hash index
- In a clustered B-Tree index records are on contiguous pages
- Bitmap is proportional to size of table and non-clustered for record access

B-Tree, Hash Tree, Bitmap





- Hash indexes don't help when evaluating range queries
- Hash index outperforms B-tree on point queries

Index Tuning Summary

- Use a hash index for point queries only
- Use a B-tree if multipoint queries or range queries are used
- Use clustering
  - if your queries need all or most of the fields of each records returned
  - if multipoint or range queries are asked
- Use a dense index to cover critical queries
  - Take a long time
  - Frequently executed
  - Have to be answered in a short time
- Don’t use an index if the time lost when inserting and updating overwhelms the time saved when querying

Τέλος Ενότητας

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημειώματα

Σημείωμα αδειοδότησης

.

Σημείωμα Αναφοράς