CSCI317 Database Performance Tuning

Index Selection Guidelines

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Index Selection Guidelines

Outline

To use index or not to use index ?

Efficiency of indexing

A wise choice of an index key

Cluster index versus non-cluster index

Hash index versus B*-tree index

Balancing the costs of index maintenance

To use an index or not to use an index ?

Main principle

- Do not build index unless some query (including the query components of updates and deletions) benefit from it

Selectivity of an attribute A in a relational table T is defined as a result of a query

Selectivity of attribute A

```
SELECT COUNT(DISTINCT A) / COUNT(*)
FROM T;
```

Selectivity of an attribute is a number in a range (0, 1]

Primary key and candidate key have the highest selectivity equal to 1

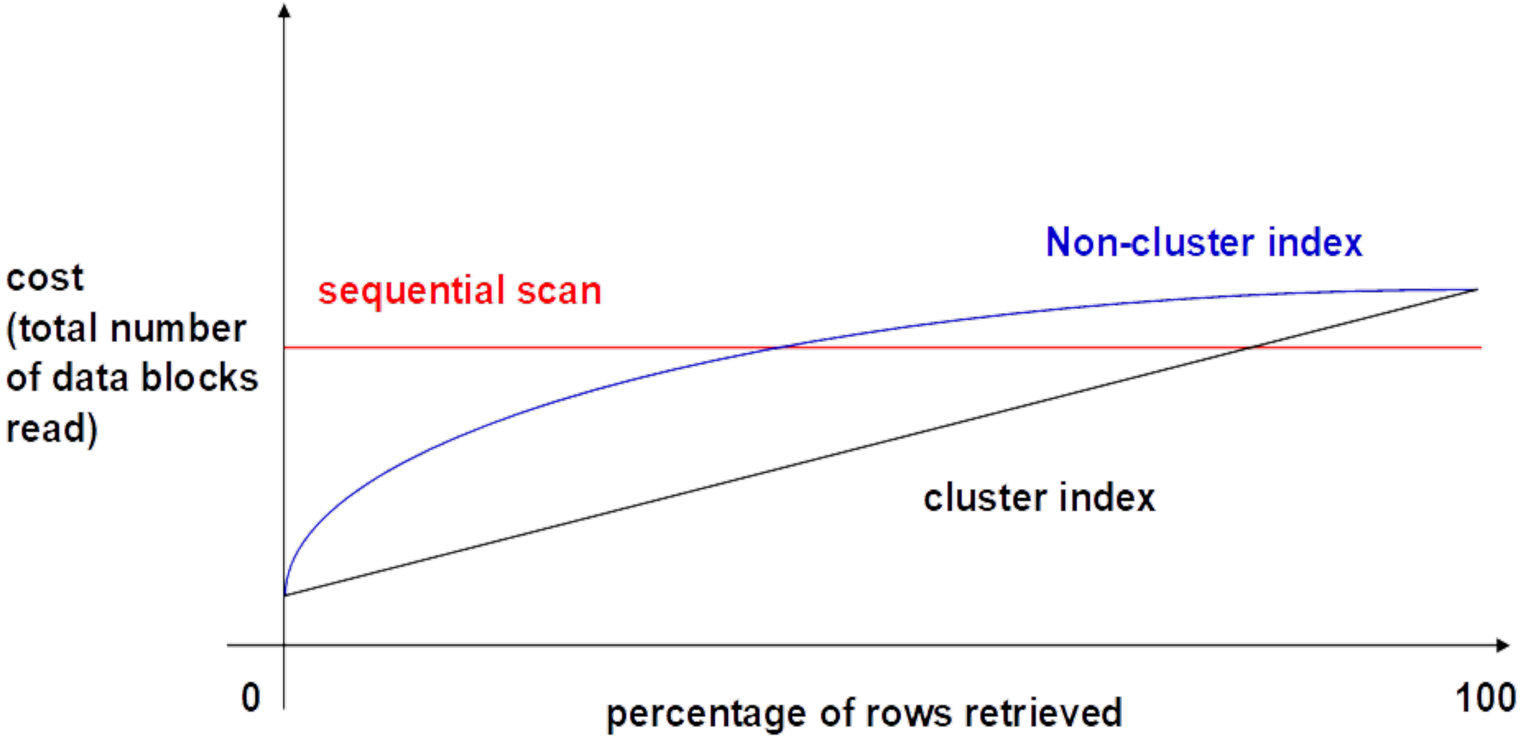
Selectivity of a set of attributes (A, B, ...) is computed in the same way a selectivity of a single attribute

Index Selection Guidelines

Outline

- To use index or not to use index ?
- Efficiency of indexing
- A wise choice of an index key
- Cluster index versus non-cluster index
- Hash index versus B*-tree index
- Balancing the costs of index maintenance

Efficiency of indexing



Index Selection Guidelines

Outline

To use index or not to use index ?

Efficiency of indexing

A wise choice of an index key

Cluster index versus non-cluster index

Hash index versus B*-tree index

Balancing the costs of index maintenance

A wise choice of an index key

An exact-match selection condition such as `attribute=value` suggests that we should consider an index on `attribute` when selectivity of `attribute` is high

If `attribute` is frequently updated or new values are added and old values are deleted then it should be B-Tree based index (majority of the cases)

If `attribute` does not change frequently then we should consider hash-based index

If a relational table that includes `attribute` changes frequently then we must use non-cluster based index

If a relational table that includes `attribute` has very infrequent changes then cluster based index is a good option

A wise choice of an index key

A range selection condition such as `attribute>value` means that we must use B-Tree based index on attribute

If a relational table that includes `attribute` changes frequently then we must use non-cluster based index

If a relational table that includes `attribute` has very infrequent changes then cluster based index is a good option

A wise choice of an index key

Index with a `composite key` should be considered when `WHERE` clause includes a condition on more than one attribute and a condition is a conjunction (`AND`) of elementary terms

For example a query like

Benefits from a composite index key

```
SELECT *
FROM EMPLOYEE
WHERE fname = 'James' AND lname = 'Bond';
```

benefits of from a composite key index `(fname, lname)`

An order of attributes in a composite key index is very important

For example a query like

May benefit from a composite index key

```
SELECT *
FROM EMPLOYEE
WHERE lname = 'Bond';
```

benefits of from a composite key index `(lname, fname)`

A wise choice of an index key

It is always worth to consider an index that allows for index only processing of a query, for example a query like

Index only processing of a query

```
SELECT fname, lname
FROM EMPLOYEE
```

does not need access to a relational table `EMPLOYEE` and entire processing can be done on a composite key index `(lname, fname)`

Another example of index only processing of a query

Index only processing of a query

```
SELECT city
FROM EMPLOYEE
WHERE city LIKE 'A%';
```

does not need access to a relational table `EMPLOYEE` and entire processing can be done on a single attribute index `(city)`

A wise choice of an index key

An index can be used for aggregation, sorting, and grouping, for example, query like

Horizontal traversal of an index

```
SELECT COUNT(*)
FROM EMPLOYEE;
```

does not need access to a relational table `EMPLOYEE` and entire processing can be done on an index on primary key

For example, a query like

Horizontal traversal of an index

```
SELECT *
FROM EMPLOYEE
ORDER BY lname
```

does not need to sort a relational table `EMPLOYEE` and sorting can be replaced with horizontal traversal through leaf level of a composite key index `(lname, fname)`

A wise choice of an index key

An index can be used for aggregation, sorting, and grouping, for example, a query like

Horizontal traversal of an index

```
SELECT lname, COUNT(*)
FROM EMPLOYEE
GROUP BY lname;
```

does not need to group a relational table `EMPLOYEE` over `lname` and entire processing can be done by a horizontal scan through leaf level of a composite key index `(lname, fname)`

Index Selection Guidelines

Outline

To use index or not to use index ?

Efficiency of indexing

A wise choice of an index key

Cluster index versus non-cluster index

Hash index versus B*-tree index

Balancing the costs of index maintenance

Cluster index versus non-cluster index

Clustering has a very positive impact on performance of query processing

Clustering has a very negative impact on performance on insert/update /delete operations

Range queries are likely to benefit from clustering

If an index enables index-only query computation strategy then such index does not need to be cluster index

Index Selection Guidelines

Outline

To use index or not to use index ?

Efficiency of indexing

A wise choice of an index key

Cluster index versus non-cluster index

Hash index versus B*-tree index

Balancing the costs of index maintenance

Hash index versus B*-Tree index

B*-Tree index is preferable when a table is frequently updated

Only B*-Tree index can be used for range queries

Hash index is preferable for equality queries on read-only relational table

Hash index better supports hash-based implementation of join operation

Index Selection Guidelines

Outline

To use index or not to use index ?

Efficiency of indexing

A wise choice of an index key

Cluster index versus non-cluster index

Hash index versus B*-tree index

Balancing the costs of index maintenance

Balancing the costs of index maintenance

If index maintenance slows down important `UPDATE`/`DELETE`/`INSERT` operations - drop an index

Index may speed up `UPDATE`/`DELETE`/`INSERT` operations when their computation involves query processing

References

Cookbook, How to measure and how to improve performance of database applications, how to choose the best index, how to analyze index structures ?

Ramakrishnan R., Gehrke J., Database Management Systems, chapters 20.1-20.3

Lightstone, S., Teorey T., Nadeau T., Physical Database Design, The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More, Morgan Kaufmann Publishers, 2007, chapter 4

Sasha, D., Bonnet, P., Database Tuning Principles, Experiments, and Troubleshooting Techniques, Morgan Kaufmann, 2003, chapter 3

Kyte, T., Expert Oracle Database Architecture, 9i and 10g Programming Techniques and Solutions, APress, 2005, chapter 11