CSCI235 Database Systems

Introduction to Indexing

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Introduction to Indexing

Outline

Index ? What is it ?

An index is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations

An index is used to efficiently retrieve all records that satisfy a search condition on the search key fields of the index

An index is a function $f : K \rightarrow \wp(id_R)$ where K is set of keys and

$\wp(id_R)$ is a powerset (a set of all sets) of identifiers (addresses) id_R of the records in a set R

Let `EMP` be a relational table over a relational schema

```
Employee(enumber, name, department)
```

Then, $F_{department}$domain(department) $\rightarrow \wp(id_{EMP})$ is a function that maps the names of departments in domain(DEPARTMENT) into the sets of identifiers of rows $\wp(id_{EMP})$ in relational table `EMP`

$F_{department}(d)$ returns the identifiers of all rows where a value of attribute `department` is equal to d

Introduction to Indexing

Outline

Index versus indexed file organization

An indexed file organization (index organized file) is a function $f : K \rightarrow \wp(R)$ where K is a set of keys and $\wp(R)$ is a powerset (a set of sets) of records R

An index maps a value into set of row identifiers

An index organized file maps a value into a set of records

A relational table can be indexed or it can be index organized

An indexed relational table consists of several index(es) created separately from implementation of a relational table itself

An index organized relational table consists only of implementation of one index where an index key is the same as a relational schema of an index organized table

Indexing in database systems is transparent to data manipulation and data retrieval operations

It means that a database system automatically modifies an index and automatically decides whether an index is used for search

Introduction to Indexing

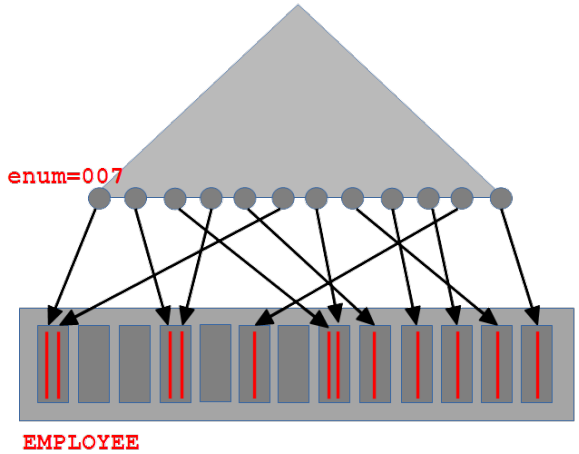
Outline

Primary (unique) index

A primary (unique) index is an index on a set of attributes equal to primary or candidate key

A primary index is a function $f : K \rightarrow id_R$ where K is a set of key values and id_R is a set of identifiers (physical addresses) of rows in a relational table R

A primary index maps and index key into a single row identifier (physical address of a row)



$F_{enum} : domain(enum) \rightarrow id_{EMPLOYEE}$

Primary (unique) index

A primary key in a relational table is always automatically indexed by a database system

For example, a relational table `EMPLOYEE` created over a relational schema `Employee(enum, name, department)` where `enum` is a primary key has an index automatically created on an attribute (`enum`)

For example, a relational table `ENROLMENT` created over a relational schema `Enrolment(snumber,code,edate)` where (`snumber,code)` is a primary key has an index automatically created on a set of attributes (`snumber,code`)

An index on (`snumber,code`) is a composite index

Introduction to Indexing

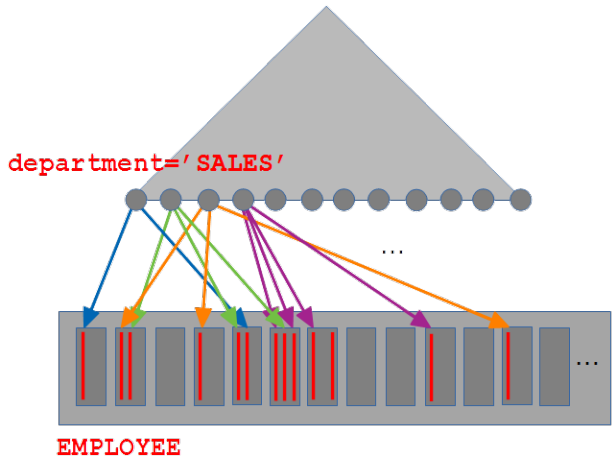
Outline

Secondary (nonunique) index

A secondary index is an index which is not primary

A secondary index is a function $f : K \rightarrow \wp(id_R)$ where K is a set of key values and id_R is a set of identifiers (physical addresses) of rows in a relational table R

A secondary index maps and index key into a set of row identifiers (a set of physical addresess of the rows)



$F_{department}$: domain(department) $\rightarrow \wp(id_{EMPLOYEE})$

In HTML view press 'p' to see the lecture notes
TOP

Secondary (nonunique) index

For example, an index on an attrtibute (`name`) in a relational table `EMPLOYEE` created over a relational schema `Employee(enum, name, department)` is a secondary (nonunique) index

For example, an index on a set of attrtibutes (`name, department`) in a relational table `EMPLOYEE` created over a relational schema `Employee(enum, name, department)` is a secondary index

For example, an index on an attribute (`snumber`) in a relational table `ENROLMENT` created over a relational schema `Enrolment(snumber,code,edate)` is a secondary index

An index on a set of attributes (`enum,name`) in a relational table `EMPLOYEE` created over a relational schema `Employee(enum, name, department)` is still a primary index because (`enum,name`) is a superkey

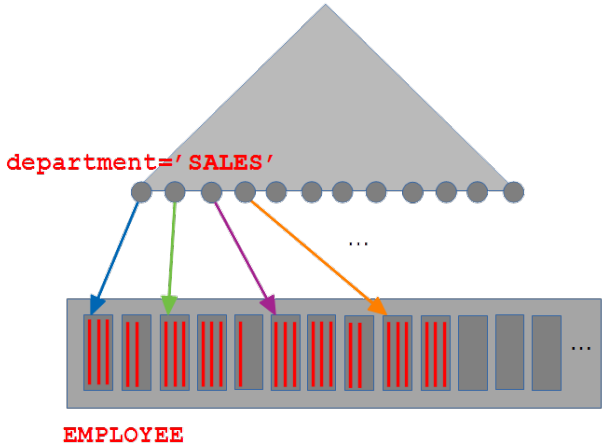
Introduction to Indexing

Outline

Clustered index

A clustered index is an index organized such that the ordering of rows is the same as ordering of keys in the index

A clustered index is a function f: K → id_R where K is a set of keys and id_R is a set of row identifiers (addresses) in arelational table R such that f(v) returns row identifier (address) of the first row in a sequence of rows such that a value of atribute K is equal to v



$f_{department}$: domain(department) → $id_{EMPLOYEE}$

In HTML view press 'p' to see the lecture notes
TOP

Clustered index

Every primary index is clustered

Clustered index provides faster access to data than nonclustered secondary index

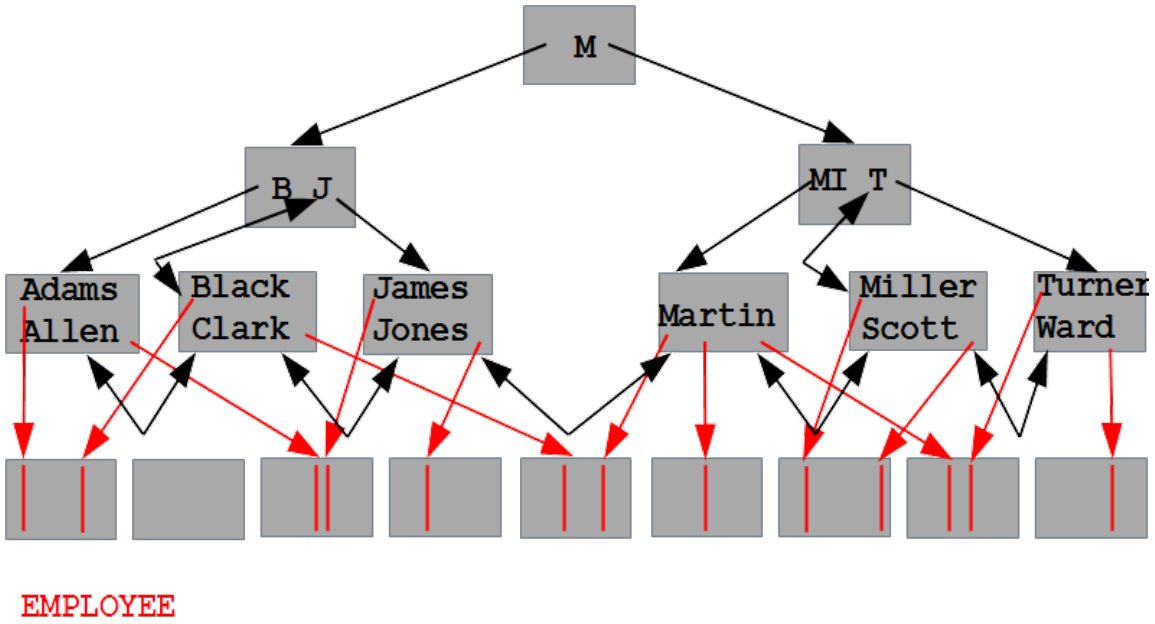
Clustered index has a very negative impact on performance of `INSERT` and `UPDATE` SQL statements

Therefore, clustered indexing should be applied to mainly to read-only data

Introduction to Indexing

Outline

B*-tree index implementation



B*-tree can be traversed either:

- vertically from root to leaf level of a tree
- horizontally either from left corner of leaf level to right corner of leaf level or the opposite
- vertically and later on horizontally either towards left lower corner or right lower corner of leaf level

In HTML view press 'p' to see the lecture notes

Introduction to Indexing

Outline

Traversals of B*-tree index

An index on a primary key (`enum`) in a relational table `EMPLOYEE` created over a relational schema
`Employee(enum, name, department, salary)`
is always built automatically by a database system

A name of an index is the same as a name of primary key constraint in a relational table `EMPLOYEE`

The following queries are processed through a vertical traversal of an index on (`enum`)

```SQL
SELECT *
FROM EMPLOYEE
WHERE enum = 007;
```

```SQL
SELECT *
FROM EMPLOYEE
WHERE enum = 007 AND department = 'MI6';
```

```SQL
SELECT enum
FROM EMPLOYEE
WHERE enum = 007;
```

In HTML view press 'p' to see the lecture notes
TOP

Traversals of B*-tree index

The following queries are processed through a horizontal traversal of leaf level of an index on (enum)

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

```sql
SELECT COUNT(enum)
FROM EMPLOYEE;
```

```sql
SELECT COUNT(name)  /* Only if name IS NOT NULL */
FROM EMPLOYEE;
```

```sql
SELECT enum
FROM EMPLOYEE;
```

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

```sql
SELECT enum
FROM EMPLOYEE
ORDER BY enum;
```

Traversals of B*-tree index

Assume that we created an index on attribute (`name`) in a relational table `EMPLOYEE` created over a relational schema `Employee(enum, name, department, salary)`

The following queries will be processed through a vertical traversal of an index on (`name`)

```SQL
SELECT *
FROM EMPLOYEE
WHERE  name = 'James';
```

```SQL
SELECT *
FROM EMPLOYEE
WHERE name = 'James' and department = 'MI6';
```

```SQL
SELECT count(*)
FROM EMPLOYEE
WHERE name = 'James'
```

Traversals of B*-tree index

Assume that we created an index on the attributes
`(name, department)`
in a relational table `EMPLOYEE` created over a relational schema
`Employee(enum, name, department, salary)`

The following queries are processed through a vertical traversal of an index on (`name, department`)

```SQL
SELECT *
FROM EMPLOYEE
WHERE name = 'James' and department = 'MI6';
```

```SQL
SELECT count(*)
FROM EMPLOYEE
WHERE name = 'James' and department = 'MI6';
```

```SQL
SELECT *
FROM EMPLOYEE
WHERE name = 'James' and department = 'MI6' and salary > 1000;
```

Traversals of B*-tree index

The following queries can be processed through a vertical traversal and later on horizontal traversal of an index on (`enum`)

```SQL
SELECT *
FROM EMPLOYEE
WHERE enum > 300;
```

```SQL
SELECT count(*)
FROM EMPLOYEE
WHERE enum < 007;
```

```SQL
SELECT *
FROM EMPLOYEE
WHERE enum > 300 and salary > 1000;
```

Traversals of B*-tree index

Assume that we created an index on the attributes
`(name, department)`
in a relational table `EMPLOYEE` created over a relational schema
`Employee(enum, name, department, salary)`

The following queries can be processed through a vertical traversal and later on horizontal traversal of an index on (`name, department`)

```SQL
SELECT *
FROM EMPLOYEE
WHERE name > 'James';
```

```SQL
SELECT count(*)
FROM EMPLOYEE
WHERE name <= 'James';
```

```SQL
SELECT *
FROM EMPLOYEE
WHERE name = 'James' and department > 'MI6';
```

Traversals of B*-tree index

Assume that we created an index on the attributes
`(name, department)`
in a relational table `EMPLOYEE` created over a relational schema
`Employee(enum, name, department, salary)`

The following queries can be processed through a vertical traversal and later on horizontal traversal of an index on (`name, department`)

```SQL
SELECT *
FROM EMPLOYEE
WHERE name > 'James' and salary > 1000;
```

```SQL
SELECT name, count(*)
FROM EMPLOYEE
WHERE name > 'James' and salary > 1000
GROUP BY name;
```

```SQL
SELECT *
FROM EMPLOYEE
WHERE name > 'James' and salary > 1000
ORDER BY name;
```

Traversals of B*-tree index

Assume that we created an index on the attributes
`(name, department)`
in a relational table `EMPLOYEE` created over a relational schema
`Employee(enum, name, department, salary)`

The following queries can be processed through a horizontal traversal of an index on (`name, department`)

```SQL
SELECT *
FROM EMPLOYEE
WHERE department = 'MI6;
```

```SQL
SELECT *
FROM EMPLOYEE
WHERE department > 'MI6;
```

```SQL
SELECT name, department
FROM EMPLOYEE;
```

```SQL
SELECT name, department, count(*)
FROM EMPLOYEE
GROUP BY name, department;
```

Introduction to Indexing

Outline

Examples

What index should be created on a relational table `DEPARTMENT` created over a relational schema
`Department(dname, chairperson, budget)`
to speed up the following queries ?

```SQL
SELECT *
FROM DEPARTMENT
WHERE dname = 'MI6';
```

There is no need for any new index because an attribute `dname` is a primary key and it is automatically indexed

```SQL
SELECT *
FROM DEPARTMENT
WHERE dname = 'MI6' AND  budget > 10000;
```

There is no need for any new index because an attribute `dname` is a primary key and it is automatically indexed

Examples

What index should be created on a relational table `DEPARTMENT` created over a relational schema
`Department(dname, chairperson, budget)`
to speed up the following queries ?

```sql
SELECT *
FROM DEPARTMENT
WHERE budget = 10000;
```

SQL

```sql
CREATE INDEX DEPT_IDX_BUDGET ON DEPARTMENT(budget);
```

SQL

```sql
SELECT *
FROM DEPARTMENT
WHERE budget = 10000 and chairperson = 'James';
```

SQL

```sql
CREATE INDEX DEPT_IDX_BC ON DEPARTMENT(budget,chairperson);
```

SQL

```sql
SELECT DISTINCT chairperson
FROM DEPARTMENT;
```

SQL

```sql
CREATE INDEX DEPT_IDX_CHAIR ON DEPARTMENT(chairperson);
```

SQL

Examples

What index should be created on a relational table `DEPARTMENT` created over a relational schema
`Department(dname, chairperson, budget)`
to speed up the following queries ?

```SQL
SELECT *
FROM DEPARTMENT
ORDER BY budget;
```

```SQL
CREATE INDEX DEPT_IDX_BUDGET ON DEPARTMENT(budget);
```

```SQL
SELECT chairperson, budget, count(*)
FROM DEPARTMENT
GROUP BY  budget, chairperson;
```

```SQL
CREATE INDEX DEPT_IDX_BC ON DEPARTMENT(budget,chairperson);
```

```SQL
SELECT chairperson, budget, count(*)
FROM DEPARTMENT
GROUP BY  chairperson, budget;
```

```SQL
CREATE INDEX DEPT_IDX_CB ON DEPARTMENT(chairperson,budget);
```

References

Elmasri R. and Navathe S. B., Fundamentals of Database Systems, Chapter 17 Indexing Structures for Files and Physical Database Design, 7th ed., The Person Education Ltd, 2017