

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 ?

Index versus indexed file organization

Primary (unique) index

Secondary (nonunique) index

Clustered index

B*-tree index implementation

Traversals of B*-tree index

Examples

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

2/30

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

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

Introduction to Indexing

Outline

[Index ? What is it ?](#)

[Index versus indexed file organization](#)

[Primary \(unique\) index](#)

[Secondary \(nonunique\) index](#)

[Clustered index](#)

[B*-tree index implementation](#)

[Traversals of B*-tree index](#)

[Examples](#)

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

4/30

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

[Index ? What is it ?](#)

[Index versus indexed file organization](#)

[Primary \(unique\) index](#)

[Secondary \(nonunique\) index](#)

[Clustered index](#)

[B*-tree index implementation](#)

[Traversals of B*-tree index](#)

[Examples](#)

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

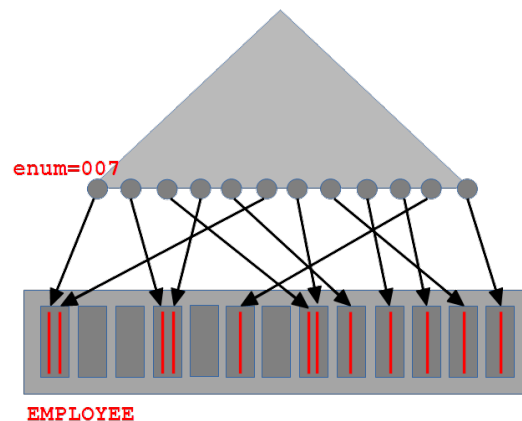
6/30

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)



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

Created by Janusz R. Getta, CSC1235 Database Systems, Spring 2020

7/30

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

[Index ? What is it ?](#)

[Index versus indexed file organization](#)

[Primary \(unique\) index](#)

[Secondary \(nonunique\) index](#)

[Clustered index](#)

[B*-tree index implementation](#)

[Traversals of B*-tree index](#)

[Examples](#)

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

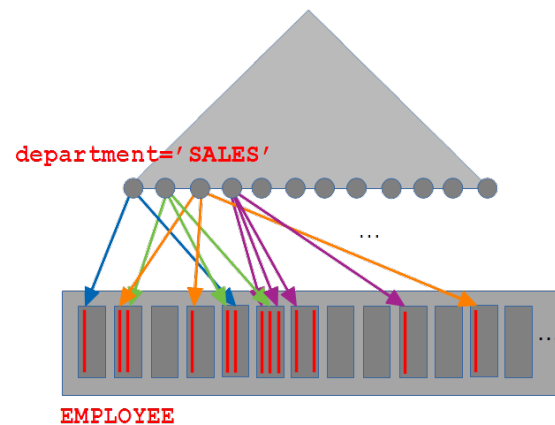
9/30

Secondary (nonunique) index

A **secondary index** is an index which is not **primary**

A **secondary index** is a function $f : K \rightarrow \wp(R)$ 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 an **index key** into a **set of row identifiers** (a set of physical addresses of the rows)



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

Secondary (nonunique) index

For example, an index on an attribute (**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 attributes (**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

[Index ? What is it ?](#)

[Index versus indexed file organization](#)

[Primary \(unique\) index](#)

[Secondary \(nonunique\) index](#)

[Clustered index](#)

[B*-tree index implementation](#)

[Traversals of B*-tree index](#)

[Examples](#)

[TOP](#)

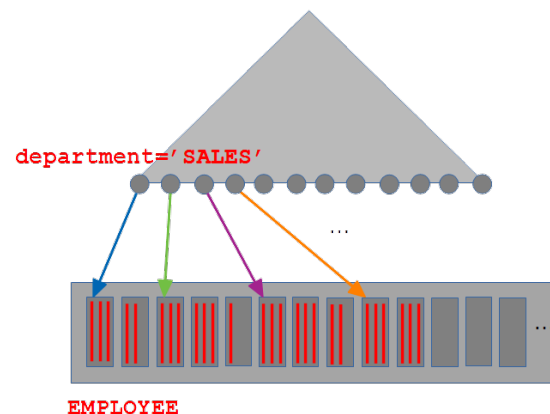
Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

12/30

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 $f: K \rightarrow id_R$ where K is a set of keys and id_R is a set of row identifiers (addresses) in a relational table R such that $f(v)$ returns row identifier (address) of the first row in a sequence of rows such that a value of attribute K is equal to v



$f_{\text{department}}: \text{domain}(\text{department}) \rightarrow id_{\text{EMPLOYEE}}$

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

[Index ? What is it ?](#)

[Index versus indexed file organization](#)

[Primary \(unique\) index](#)

[Secondary \(nonunique\) index](#)

[Clustered index](#)

[B*-tree index implementation](#)

[Traversals of B*-tree index](#)

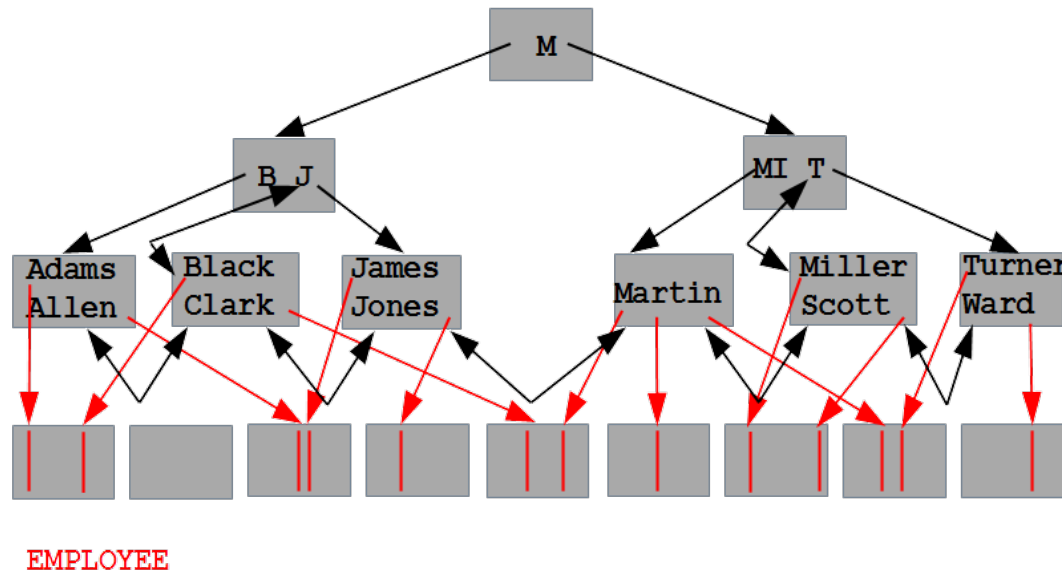
[Examples](#)

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

15/30

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

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

16/30

Introduction to Indexing

Outline

[Index ? What is it ?](#)

[Index versus indexed file organization](#)

[Primary \(unique\) index](#)

[Secondary \(nonunique\) index](#)

[Clustered index](#)

[B*-tree index implementation](#)

[Traversals of B*-tree index](#)

[Examples](#)

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

17/30

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`)

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

[SQL](#)

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

[SQL](#)

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

[SQL](#)[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

18/30

Traversals of B*-tree index

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

```
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;
```

SQL

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**)

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

[SQL](#)

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

[SQL](#)

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

[SQL](#)

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**)

```
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;
```

[SQL](#)

Traversals of B*-tree index

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

```
SELECT *  
FROM EMPLOYEE  
WHERE snum > 300;
```

[SQL](#)

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

[SQL](#)

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

[SQL](#)

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**)

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

[SQL](#)

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

[SQL](#)

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

[SQL](#)

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**)

```
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;
```

[SQL](#)[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

24/30

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**)

```
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,
```

[SQL](#)[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

25/30

Introduction to Indexing

Outline

[Index ? What is it ?](#)

[Index versus indexed file organization](#)

[Primary \(unique\) index](#)

[Secondary \(nonunique\) index](#)

[Clustered index](#)

[B*-tree index implementation](#)

[Traversals of B*-tree index](#)

[Examples](#)

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

26/30

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 ?

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

[SQL](#)

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

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

[SQL](#)

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 ?

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

[SQL](#)

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

[SQL](#)

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

[SQL](#)

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

[SQL](#)

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

[SQL](#)

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

[SQL](#)[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

28/30

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 ?

```
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);
```

[SQL](#)[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Spring 2020

29/30

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