CSCI235 – Database Systems

Introduction to Indexing

sjapit@uow.edu.au

1 October 2021

Acknowledgements

The following presentation were adapted from the lecture slides of:

CSCI235 – Database Systems, 06Introduction to Indexing

By Dr Janusz R. Getta, University of Wollongong, Australia

Spring, 2018

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

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 $\boldsymbol{f}$: $\boldsymbol{K} \rightarrow \wp(\boldsymbol{id_R})$ where $\boldsymbol{K}$ is set of keys and $\wp(\boldsymbol{id_R})$ is a powerset (a set of all sets) of identifiers (addresses) $\boldsymbol{id_R}$ of the records in a set $\boldsymbol{R}$

Index? What is it?

Let $\boldsymbol{EMP}$ be a relational table over a relational schema $\boldsymbol{Employee(enumber, name, department)}$

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 $\boldsymbol{EMP}$

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

Introduction to Indexing

Index versus indexed file organization

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

Index versus indexed file organization

- 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

Primary (unique) index

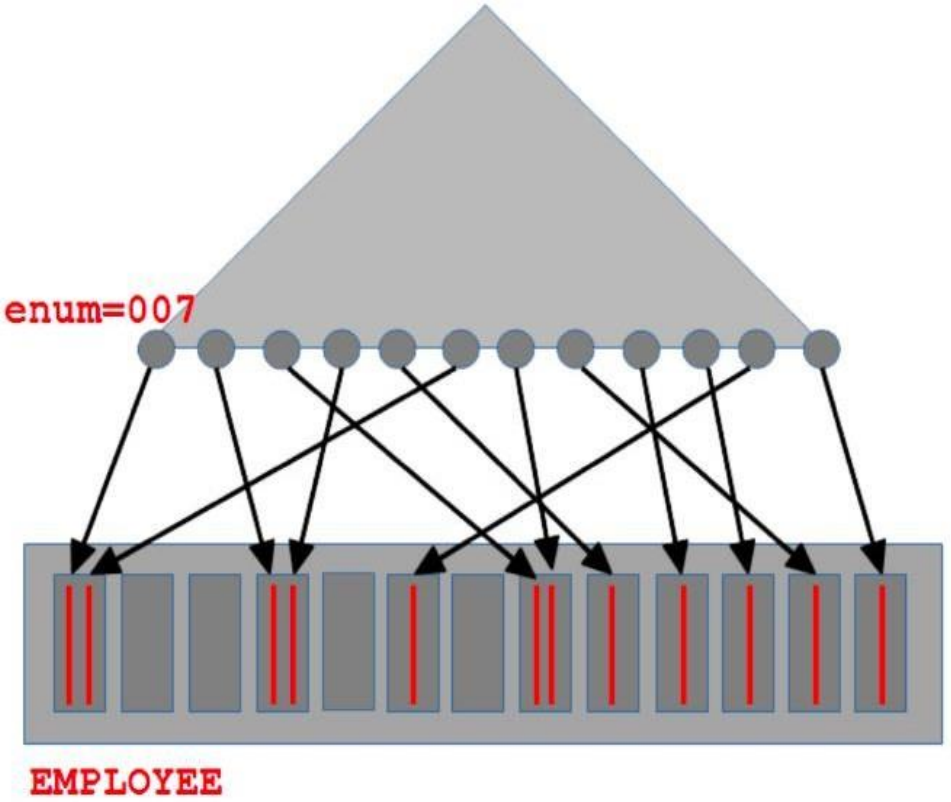
Primary (unique) index

- A **primary (unique) index** is an index on a set of attributes equal to **primary** or **candidate key**
- A **primary index** is a function $\boldsymbol{f: K \rightarrow id_R}$ where $\boldsymbol{K}$ is a set of **key values** and $\boldsymbol{id_R}$ is a set of identifiers (physical addresses) of rows in a relational table $\boldsymbol{R}$
- Primary index maps and index key into a single row identifier (physical address of a row)
- A **primary key** in a **relational table** is always **automatically indexed** by a database system

Primary (unique) index

- 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**

Primary (unique) index



F_{enum}: domain(enum) → $id_{EMPLOYEE}$

Introduction to Indexing

Secondary (non-unique) index

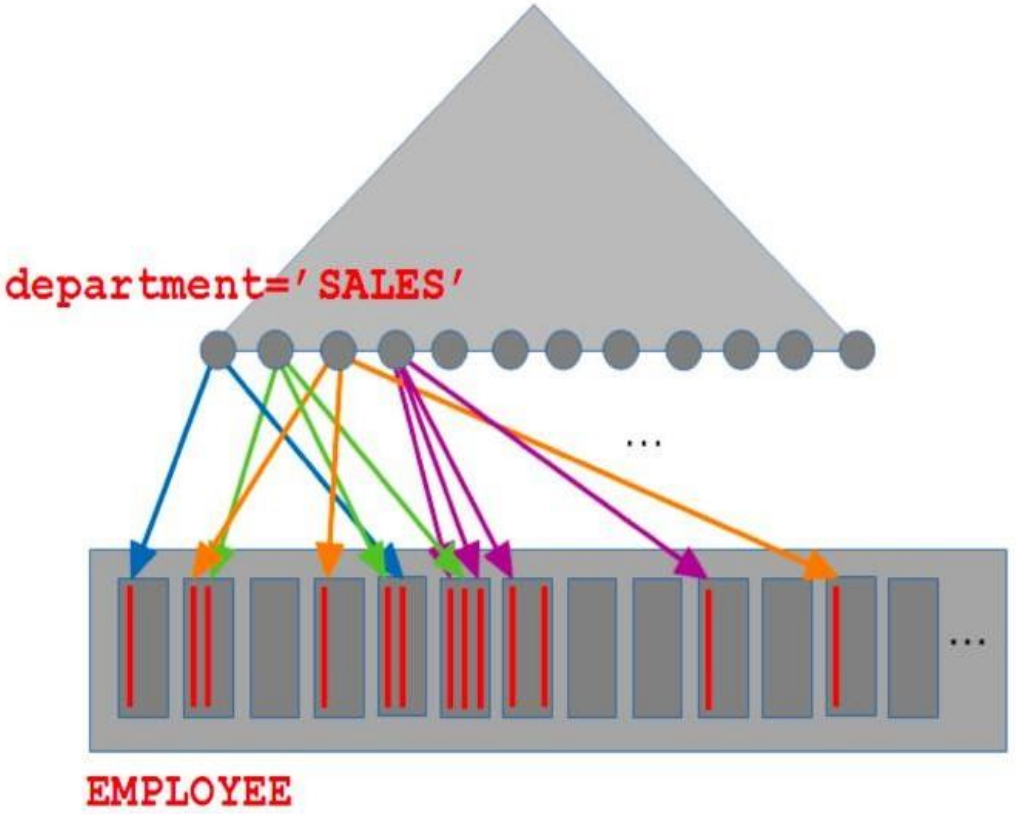
Secondary (non-unique) index

- A **secondary index** is an index which is not **primary**
- For example, an index on an attribute **(name)** in a relational table **EMPLOYEE** created over a relational schema $\boldsymbol{Employee(enum, name, department)}$ 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 $\boldsymbol{Employee(enum, name, department)}$ is a **secondary index**

Secondary (non-unique) 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 $\boldsymbol{Employee(enum, name, department)}$ is still a **primary index** because **(enum, name)** is a **superkey**

Secondary (non-unique) index



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

Introduction to Indexing

Clustered index

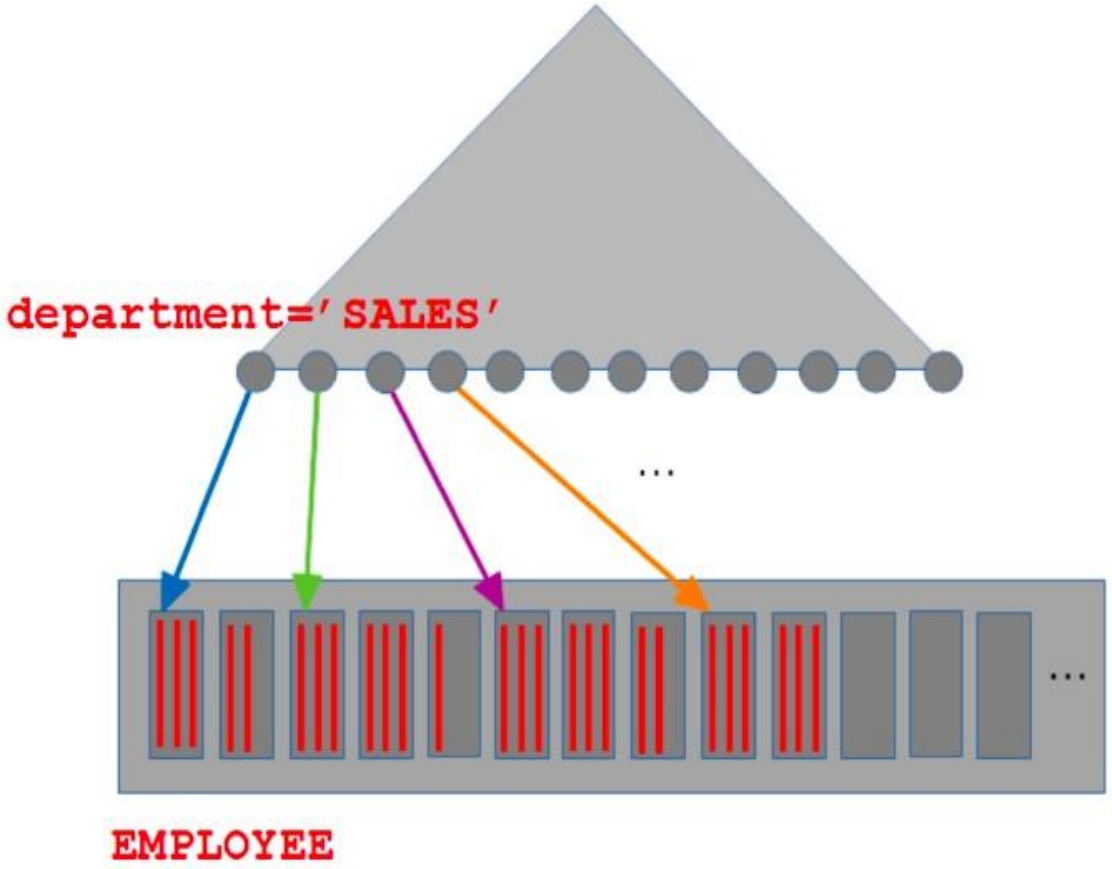
Clustered index

- 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 $\boldsymbol{f}: \boldsymbol{K} \rightarrow \boldsymbol{id_R}$ where $\boldsymbol{K}$ is a set of keys and $\boldsymbol{id_R}$ is a set of row identifiers (addresses) in a relational table $\boldsymbol{R}$ such that $\boldsymbol{f(v)}$ returns row identifier (address) of the first row in a sequence of rows such that a value of attribute $\boldsymbol{K}$ is equal to $\boldsymbol{v}$
- Every **primary index** is **clustered**
- **Clustered index** provides faster access to data than **nonclustered secondary** index

Clustered 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**

Clustered index

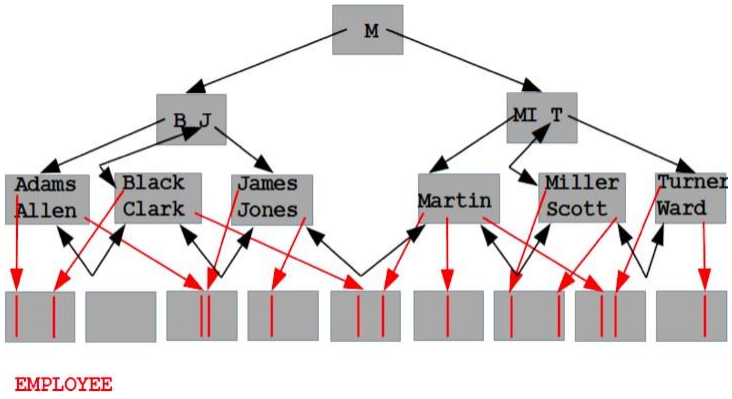


$f_{department}$: domain(department) → $id_{EMPLOYEE}$

Introduction to Indexing

B*-tree index implementation

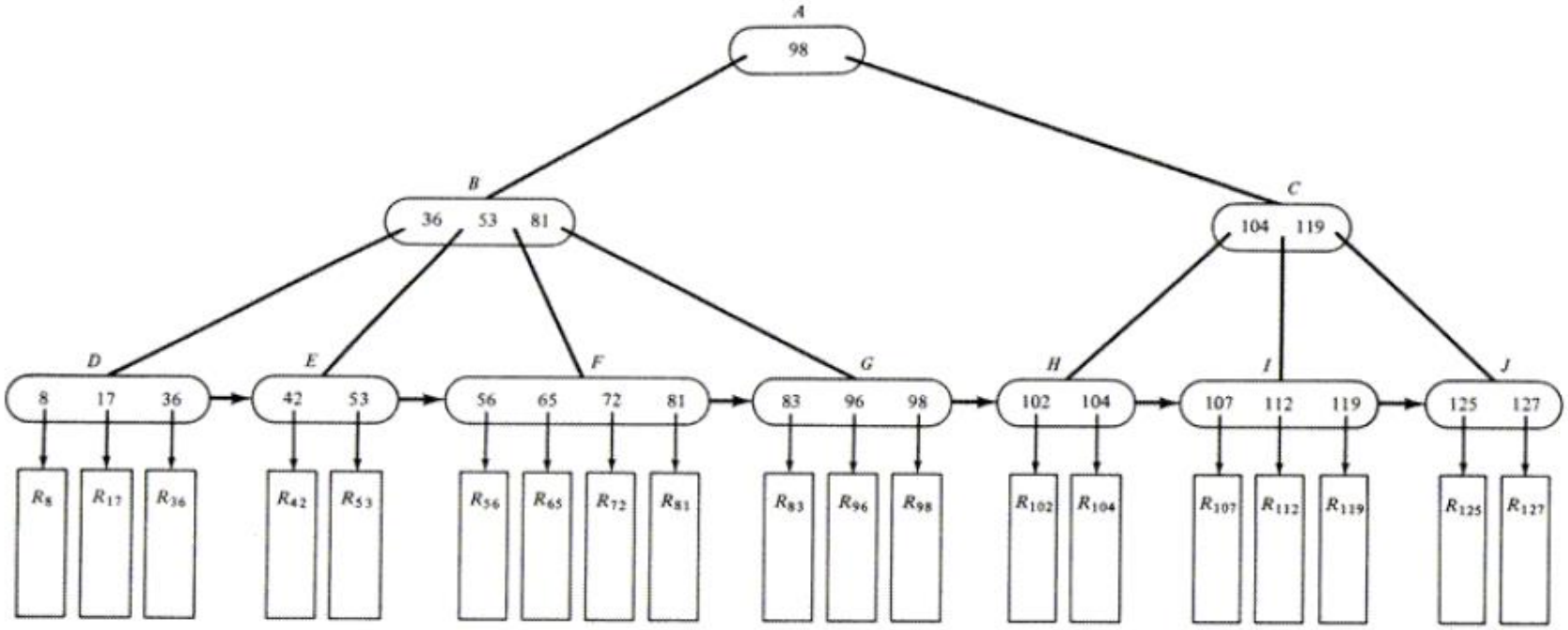
B*-tree index implementation



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**

B*-TREE


A
98
B
36 53 81
C
104 119
D
8 17 36
E
42 53
F
56 65 72 81
G
83 96 98
H
102 104
I
107 112 119
J
125 127
R_8 R_{17} R_{36} R_{42} R_{53} R_{56} R_{65} R_{72} R_{81} R_{83} R_{96} R_{98} R_{102} R_{104} R_{107} R_{112} R_{119} R_{125} R_{127}

Introduction to Indexing

Traversals of B*-tree index

Traversals of B*-tree index

- An index on a primary key **(enum)** in a relational table **EMPLOYEE** created over a relational schema $\boldsymbol{Employee(enum, name, department, salary)}$ 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)**

Traversals of B*-tree index

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

SELECT   *
FROM     EMPLOYEE
WHERE    enum=007 AND department='mi6';

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

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;

SELECT    count(enum)
FROM      EMPLOYEE;

SELECT    count(enum)   /* Only if name is NOT NULL */
FROM      EMPLOYEE;
```

Traversals of B*-tree index

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

```
SELECT      enum
FROM        EMPLOYEE;

SELECT      enum, count(*)
FROM        EMPLOYEE
GROUP BY enum;

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

$$\boldsymbol{Employee(enum, name, department, salary)}$$

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

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

Traversals of B*-tree index

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

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

Traversals of B*-tree index

Assume that we created an index on attribute **(name, department)** 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, department)**

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

Traversals of B*-tree index

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

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 (snum).

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

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

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 AND salary>1000;
```

Traversals of B*-tree index

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

$$\boldsymbol{Employee(enum, name, department, salary)}$$

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';
```

Traversals of B*-tree index

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

SELECT    *
FROM      EMPLOYEE
WHERE     name=‘James’ AND department>’MI6’;
```

Traversals of B*-tree index

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

$$\boldsymbol{Employee(enum, name, department, salary)}$$

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;
```

Traversals of B*-tree index

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

SELECT    *
FROM      EMPLOYEE
WHERE     name='James' AND salary>1000
ORDER BY name;
```

Traversals of B*-tree index

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

$$\boldsymbol{Employee(enum, name, department, salary)}$$

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

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

Traversals of B*-tree index

```
SELECT   *
FROM     EMPLOYEE
WHERE    department > ‘MI6’;

SELECT   name, department
FROM     EMPLOYEE;


SELECT    name, department, count(*)  F
ROM       EMPLOYEE
GROUP BY name, department;
```

Introduction to Indexing

Sample indexing

Sample indexing

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

```
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.

Sample indexing

```
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.

Sample indexing

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;
```

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

Sample indexing

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

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

Sample indexing

```
SELECT      DISTINCT chairperson
FROM        DEPARTMENT;

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

Sample indexing

```
SELECT      *
FROM        DEPARTMENT
ORDER BY budget;

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

Sample indexing

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

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

Sample indexing

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

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