UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

10

Intro to Indexing + SQL Indexing

CSF2600700 - BASIS DATA

Acknowledgements

This slide is a modification to supplementary slide of "Database System", 7th edition, Elmasri/Navathe, 2015: **Chapter 17 Indexing Structures for Files and Physical Database Design** used in "Basis Data" course in academic years 2018/2019 in the Faculty of Computer Science, Universitas Indonesia.

Outline

1. Intro to Index
2. Types of Index
3. Choosing Index
4. SQL Indexing

Index

Indexing is a way of sorting a number of records on multiple fields/attributes.

- ➔ Indexes are special **lookup tables** that the database search engine can use to **speed up data retrieval**, without reading the whole table.
- ➔ Simply, an index is a pointer to data in a table.
- ➔ They are based upon one or more columns but stored as a separate entity.

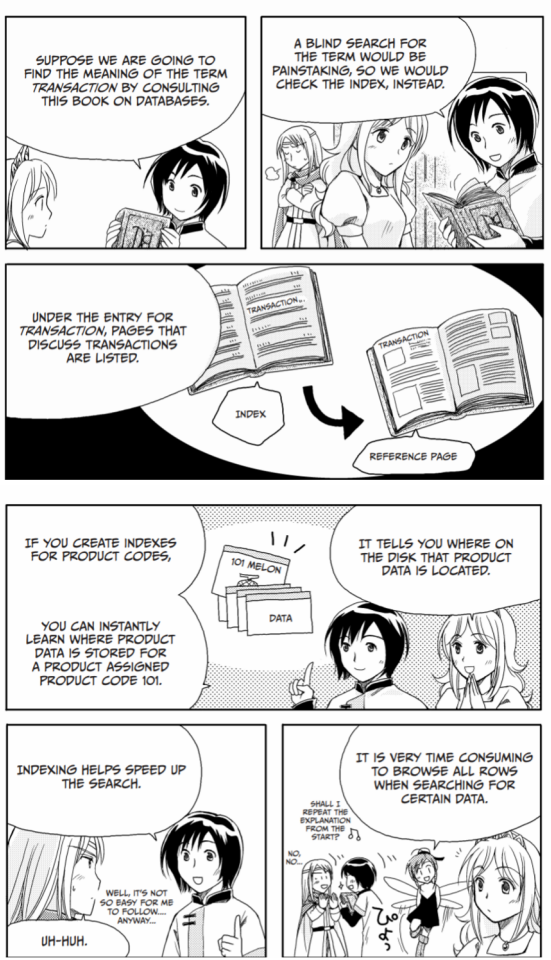
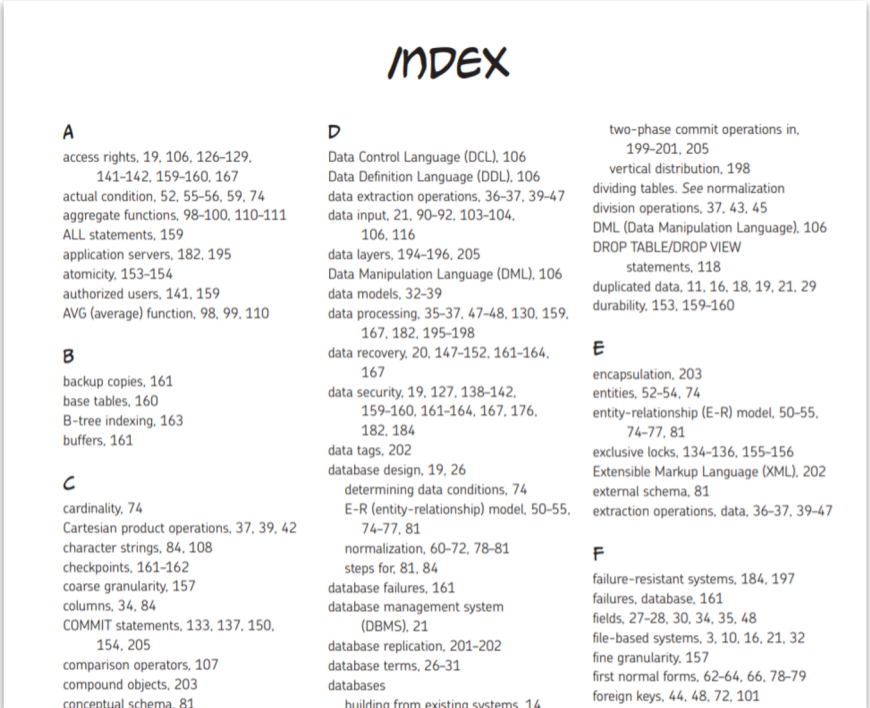


Illustration: Takahashi & Azuma (2014)

Index (Cntd.)

An index in a database is very similar to an index in the back of a book.



INDEX

A

access rights, 19, 106, 126–129, 141–142, 159–160, 167
actual condition, 52, 55–56, 59, 74
aggregate functions, 98–100, 110–111
ALL statements, 159
application servers, 182, 195
atomicity, 153–154
authorized users, 141, 159
AVG (average) function, 98, 99, 110

B

backup copies, 161
base tables, 160
B-tree indexing, 163
buffers, 161

C

cardinality, 74
Cartesian product operations, 37, 39, 42
character strings, 84, 108
checkpoints, 161–162
coarse granularity, 157
columns, 34, 84
COMMIT statements, 133, 137, 150, 154, 205
comparison operators, 107
compound objects, 203
conceptual schema, 81

D

Data Control Language (DCL), 106
Data Definition Language (DDL), 106
data extraction operations, 36–37, 39–47
data input, 21, 90–92, 103–104, 106, 116
data layers, 194–196, 205
Data Manipulation Language (DML), 106
data models, 32–39
data processing, 35–37, 47–48, 130, 159, 167, 182, 195–198
data recovery, 20, 147–152, 161–164, 167
data security, 19, 127, 138–142, 159–160, 161–164, 167, 176, 182, 184
data tags, 202
database design, 19, 26
  determining data conditions, 74
  E-R (entity-relationship) model, 50–55, 74–77, 81
  normalization, 60–72, 78–81
  steps for, 81, 84
database failures, 161
database management system (DBMS), 21
database replication, 201–202
database terms, 26–31
databases
  building from existing systems, 14
  two-phase commit operations in, 199–201, 205
  vertical distribution, 198
dividing tables. *See* normalization
division operations, 37, 43, 45
DML (Data Manipulation Language), 106
DROP TABLE/DROP VIEW statements, 118
duplicated data, 11, 16, 18, 19, 21, 29
durability, 153, 159–160

E

encapsulation, 203
entities, 52–54, 74
entity-relationship (E-R) model, 50–55, 74–77, 81
exclusive locks, 134–136, 155–156
Extensible Markup Language (XML), 202
external schema, 81
extraction operations, data, 36–37, 39–47

F

failure-resistant systems, 184, 197
failures, database, 161
fields, 27–28, 30, 34, 35, 48
file-based systems, 3, 10, 16, 21, 32
fine granularity, 157
first normal forms, 62–64, 66, 78–79
foreign keys, 44, 48, 72, 101

Index (Cntd.)

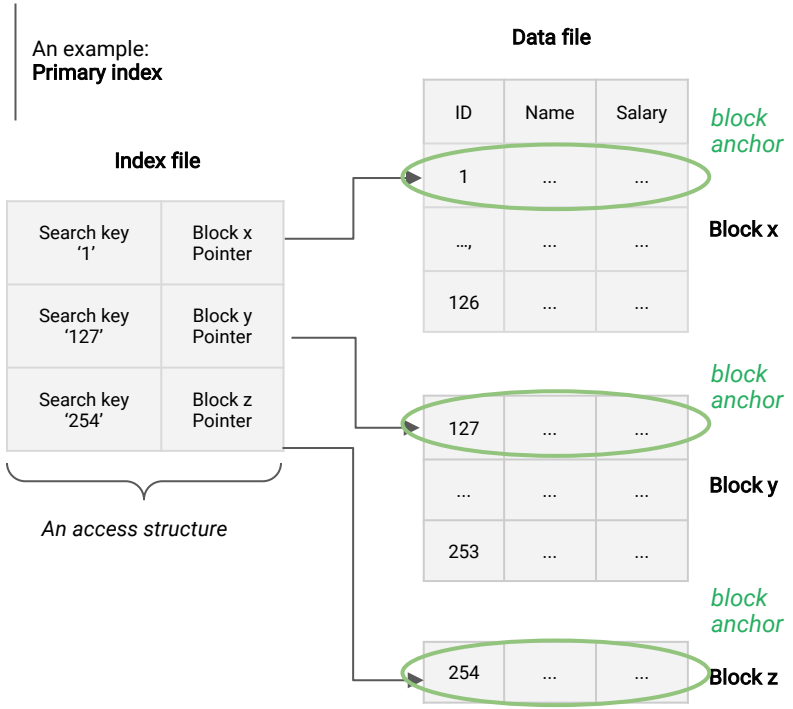
An index file consists of records (called index entries) of the form

Search key	Pointer

Search Key: **attribute** to set of attributes used to look up records in a file.

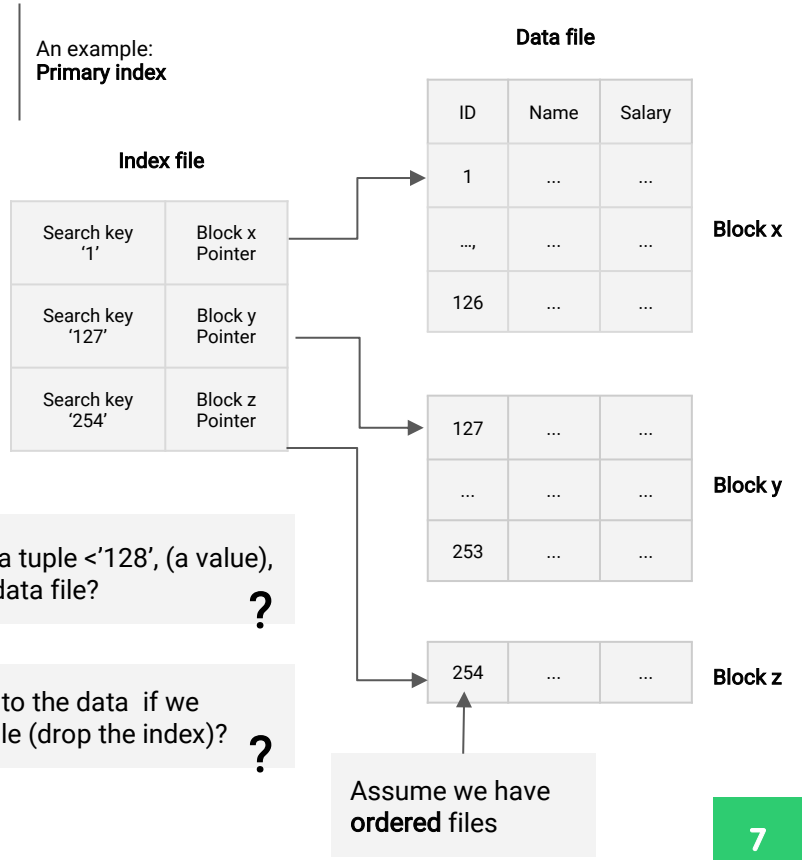
Pointer: pointer to the record or the block

- ➔ Index files are typically much smaller than the original file. Why?
- ➔ An index is an auxiliary file that makes it more efficient to search for a record in the data file



Index (Cntd.)

- An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Why?
- **Indexes can be created or dropped with no effect on the data.**



Outline

1. Intro to Index
2. Types of Index
3. Choosing Index
4. SQL Indexing

Types of Index

Primary Indexes vs. Secondary Indexes

Primary index:
in a sorted organization file, the index whose search key is the **same as the sequential order of the file**.

Secondary index:
an index whose search key is **different from the sequential order of the file**. Secondary indexes provide a mechanism for specifying an additional key for a base relation that can be used to retrieve data more efficiently

Types of Index (Cntd.)

Sparse vs. **Dense Indexes**

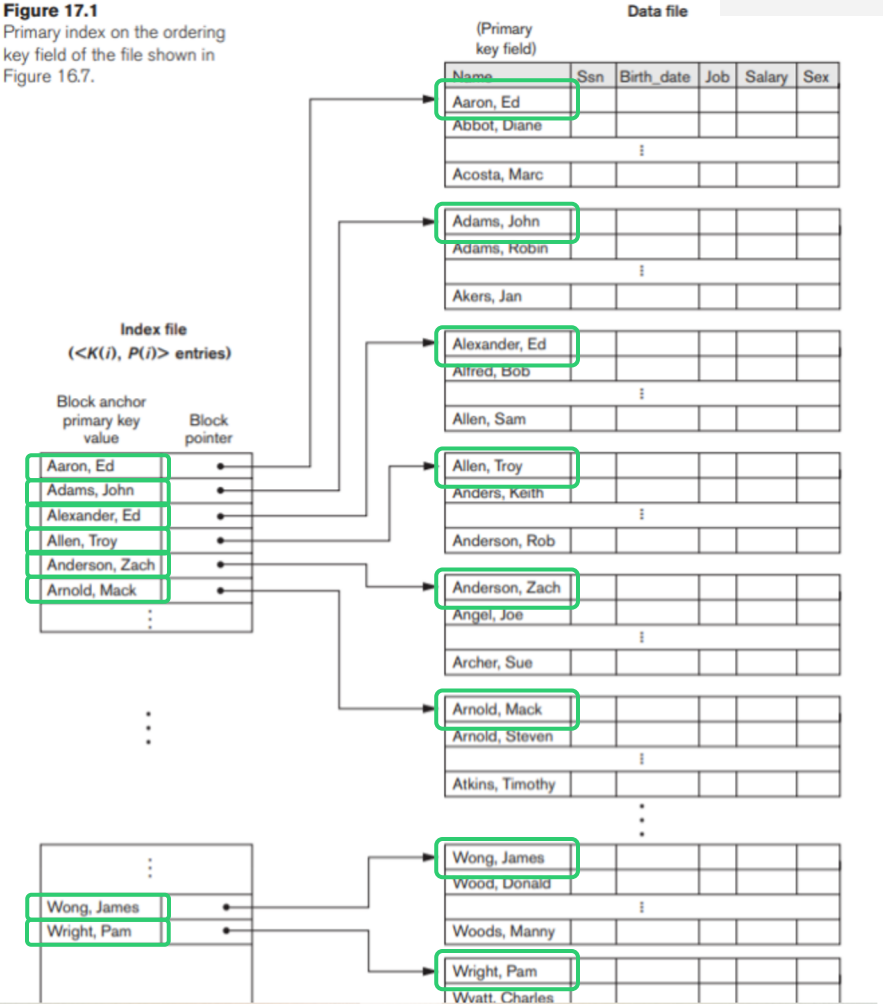
Sparse index:
index on attribute for sorting the file, only one index entry **per block** of data record.

Dense index:
one index entry **per data record**.

Example of Index

Primary, Sparse
Index

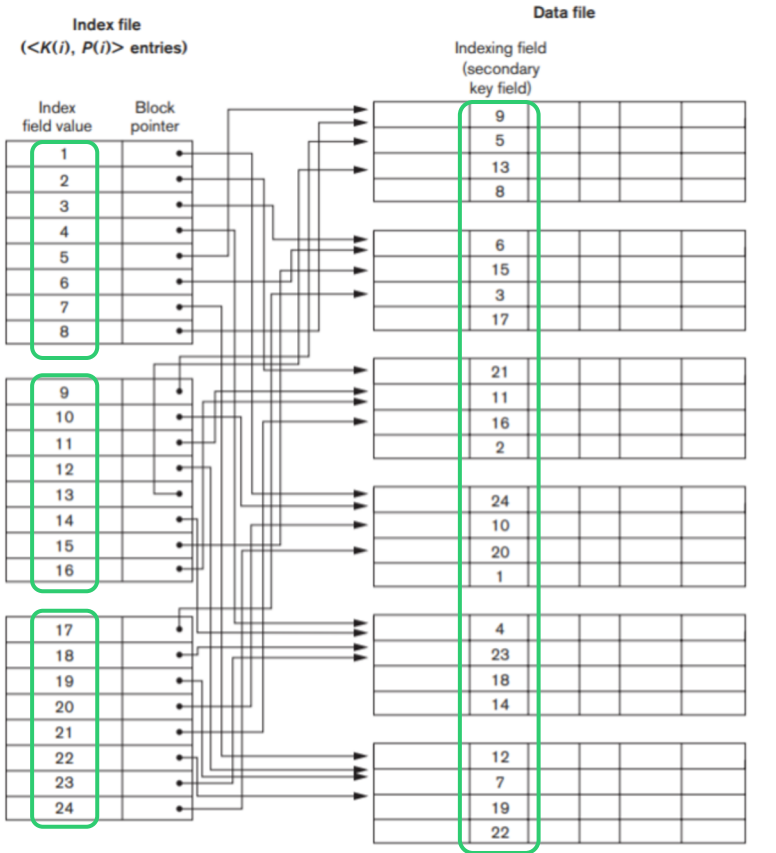
Figure 17.1
Primary index on the ordering key field of the file shown in Figure 16.7.



Example of Index (Cntd.)

Secondary, Dense
Index

Figure 17.4
A dense secondary index (with block pointers) on a nonordering key field of a file.



Types of Index (Cntd.)

Clustering Indexes

- If the ordering field is **not UNIQUE** or **non-key**, such as DEPT_NUMBER in EMP table, we can create a clustering index.
- In the index, one index entry for each distinct value

Example of Index (Cntd.)

Clustering Index

with separate **block cluster** for each group of records that share the same value for the clustering field

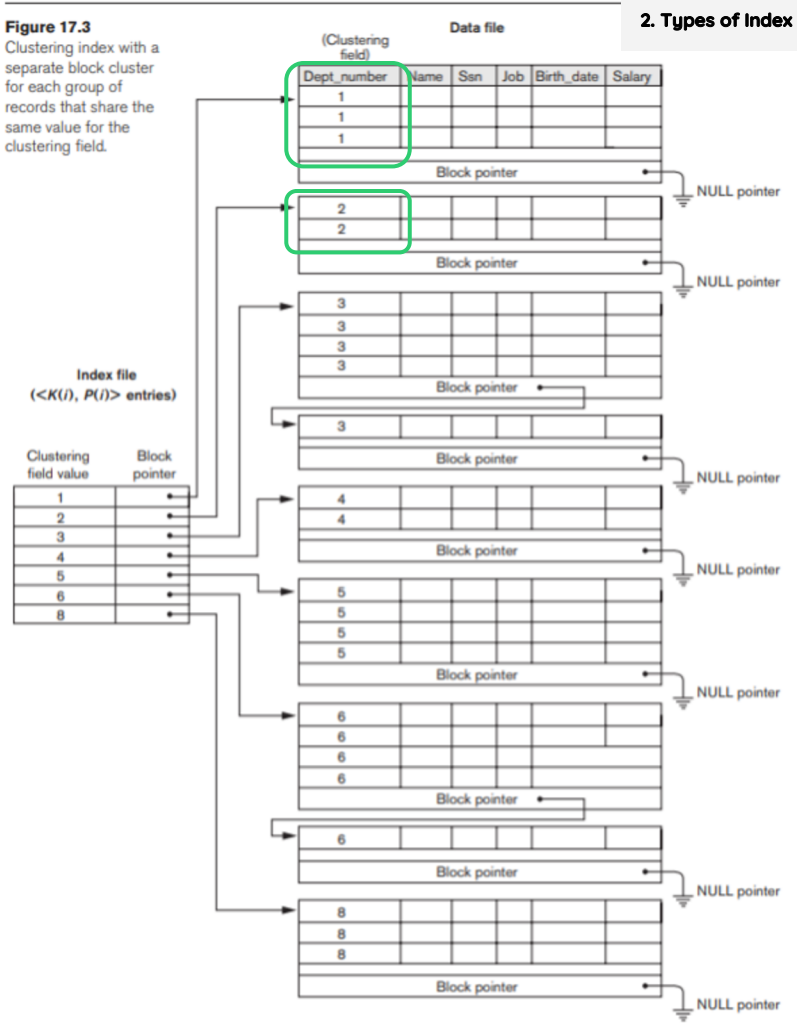


Figure 17.3
Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

Types of Index (Cntd.)

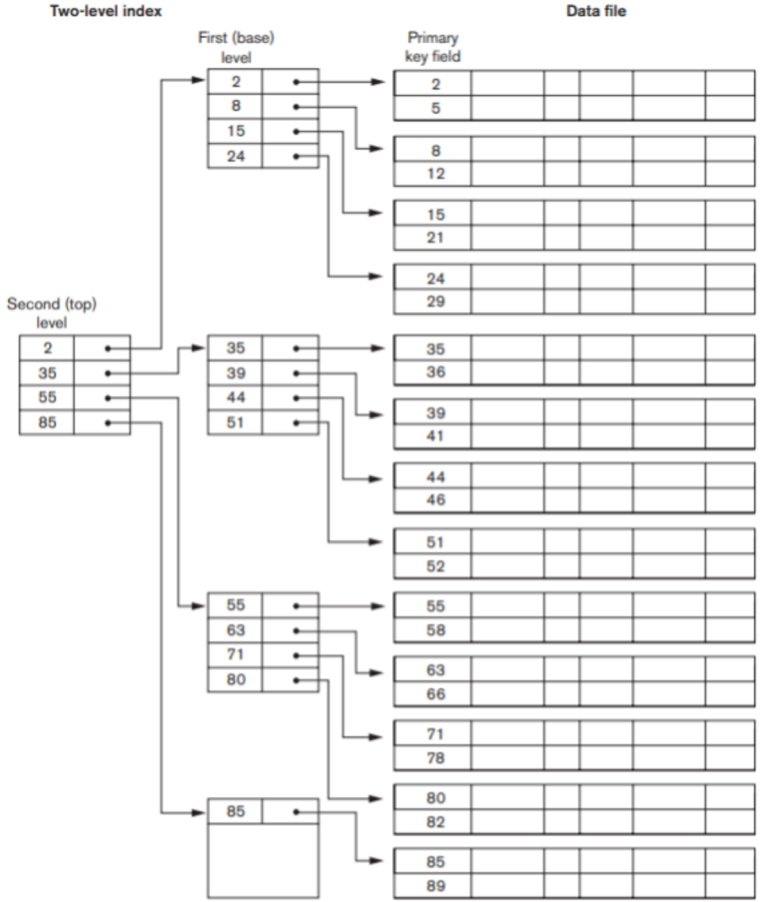
Single-Level vs. **Multi-Level Indexes**

One index per file	Multiple index per file. Index of index.

Example of Index (Cntd.)

Two-level, Primary
Index

Figure 17.6
A two-level primary index resembling ISAM (indexed sequential access method) organization.



Types of Index: Summary

Table 17.1 Types of Indexes Based on the Properties of the Indexing Field

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Table 17.2 Properties of Index Types

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

Outline

1. Intro to Index
2. Types of Index
3. Choosing Index
4. SQL Indexing

Choose indexes – Guidelines for choosing ‘wish-list’

1. Do not index **small** relations.

2. Index **PK** of a relation if it is not a key of the file organization.

3. Add **secondary index to a FK** if it is frequently accessed.

4. Add **secondary index to any attribute heavily used** as a secondary key.

5. Add secondary index on **attributes involved in: selection or join criteria**; ORDER BY; GROUP BY; and other operations involving sorting (such as UNION or DISTINCT).

6. Add secondary index on attributes **involved in built-in functions**.

7. **Avoid indexing** an attribute or relation that is frequently **updated**.

8. **Avoid indexing** an attribute if the query will retrieve **a significant proportion of the relation**.

9. **Avoid indexing** attributes that consist of **long character strings**.

Outline

1. Intro to Index
2. Types of Index
3. Choosing Index
4. SQL Indexing

CREATE INDEX

Creating an index involves the CREATE INDEX statement,

- ➔ Which allows you to name the index,
- ➔ To specify the table and which column(s) to index,
- ➔ And to indicate whether the index is in ascending or descending order

The basic syntax:

```
CREATE INDEX index_name ON table_name (column name);
```

```
CREATE INDEX Emp_idx ON EMPLOYEE (Ssn);
```

CREATE INDEX (Cntd.)

Multiple column index

```
CREATE INDEX index_name ON table_name (column_name1, column_name2);
```

```
CREATE INDEX WORKS_ON_Idx ON WORKS_ON (pno, hours);
```

Which column to choose is based on frequently queried column in `WHERE` clause

DROP INDEX

```
DROP INDEX index_name;
```

Care should be taken when dropping an index because performance may be slowed or improved

When Not to Use Index

Although indexes are intended to enhance a database's performance, there are times when they should be avoided. The following guidelines indicate when the use of an index should be reconsidered:

- ➔ Indexes should not be used on **small tables**.
- ➔ Tables that have **frequent, large batch update or insert operations**.
- ➔ Indexes should not be used on columns that contain **a high number of NULL values**.
- ➔ Columns that are **frequently manipulated** should not be indexed.

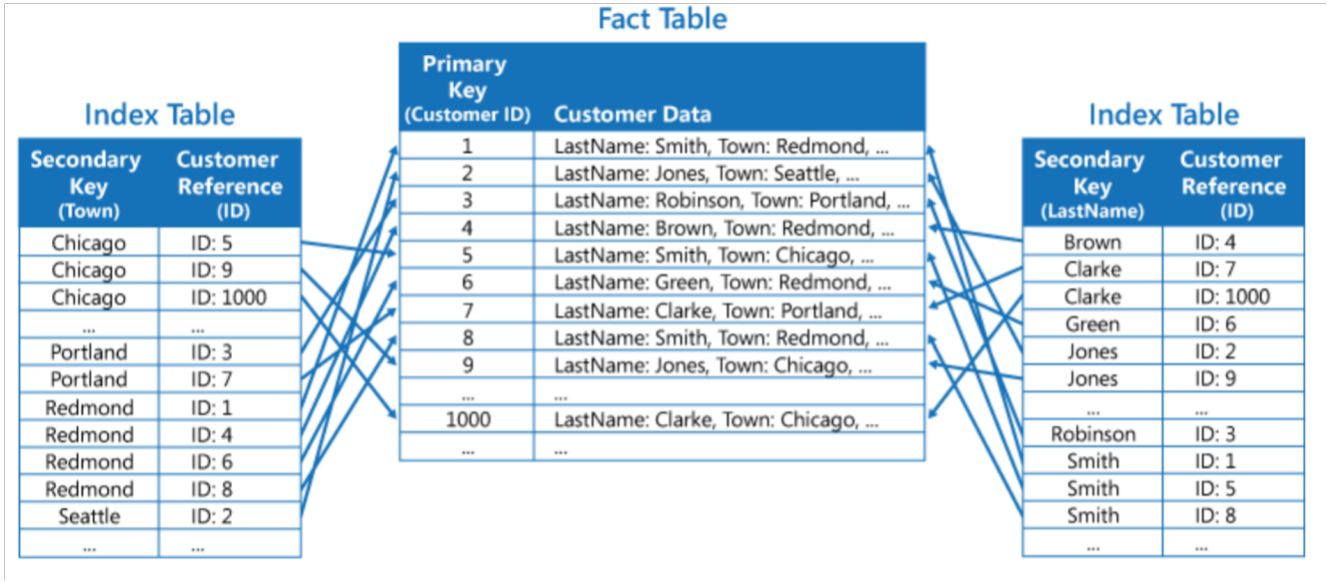
Latihan (1)

1. Mengapa kita perlu menghindari pembuatan index pada kolom yang sering dilakukan operasi UPDATE atau INSERT?

2. Jika kita menghapus index, maka data pada kolom yang indexnya dihapus juga akan ikut terhapus. Benar atau salah pernyataan ini? Berikan alasannya

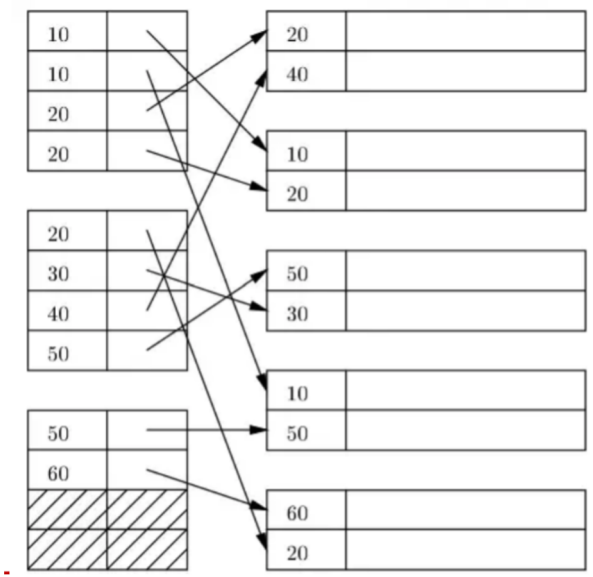
Latihan (2)

Apa **tipe index** dari gambar ini? Jelaskan alasannya



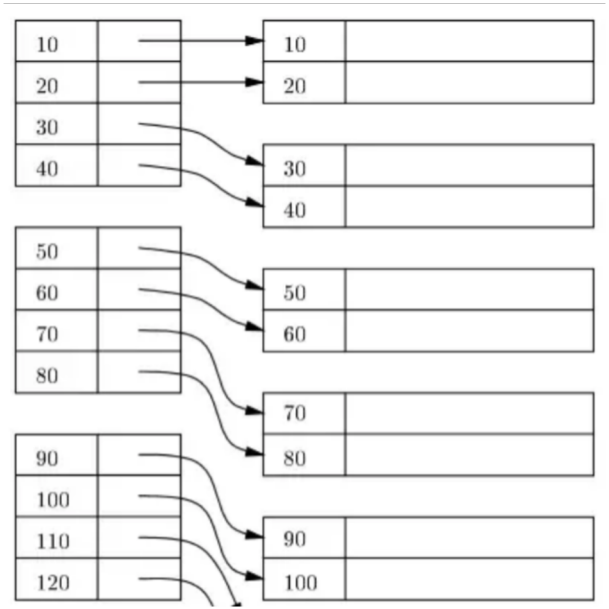
Latihan (3)

Apa **tipe index** dari gambar ini? Jelaskan alasannya



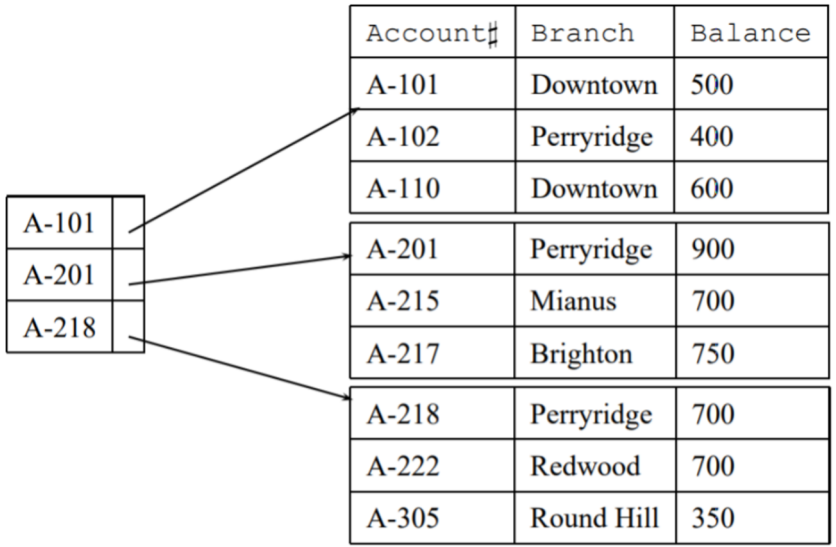
Latihan (4)

Apa **tipe index** dari gambar ini? Jelaskan alasannya



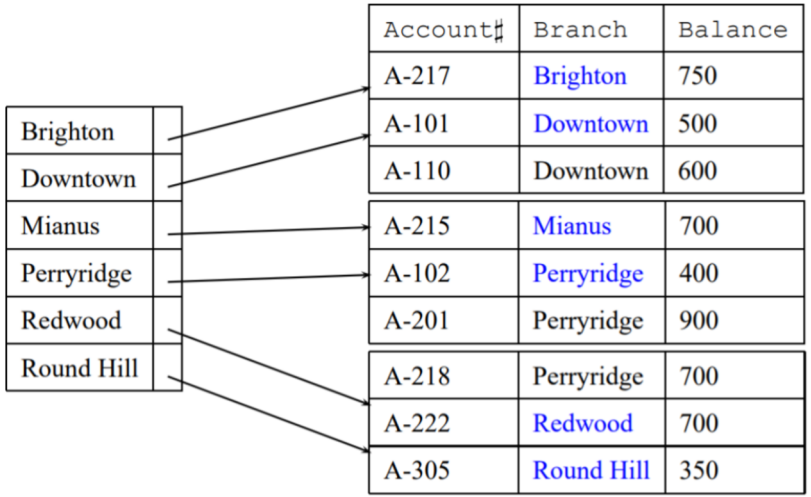
Latihan (5)

Apa **tipe index** dari gambar ini? Jelaskan alasannya



Latihan (6)

Apa **tipe index** dari gambar ini? Jelaskan alasannya



Latihan (7)

Sebuah tabel bernama DEPARTMENT tersusun dari tiga kolom yaitu Dno, Dname dan DLocation.

Kolom/atribut yang sering digunakan sebagai query statement pada tabel DEPARTMENT tersebut adalah Dno dan DLocation.

Buatlah sebuah **SQL statement** untuk membuat index bernama department_idx yang dibuat pada tabel DEPARTMENT tersebut.

Q&A