

IF3140 – Sistem Basis Data

Introduction to

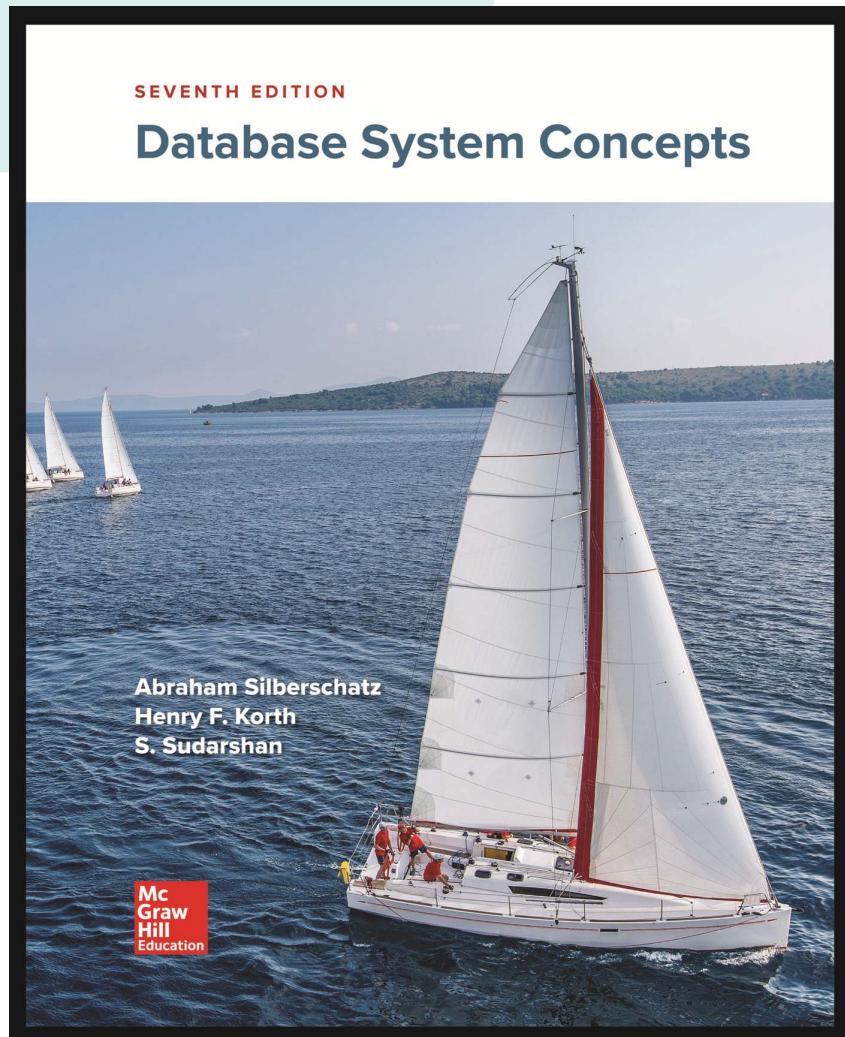
Database System

SEMESTER I TAHUN AJARAN 2024/2025



KNOWLEDGE & SOFTWARE ENGINEERING





References

Abraham Silberschatz, Henry F. Korth, S. Sudarshan :
"Database System Concepts",
7th Edition

- Chapter 1.6 : Database Engine
- Chapter 15 : Query Processing

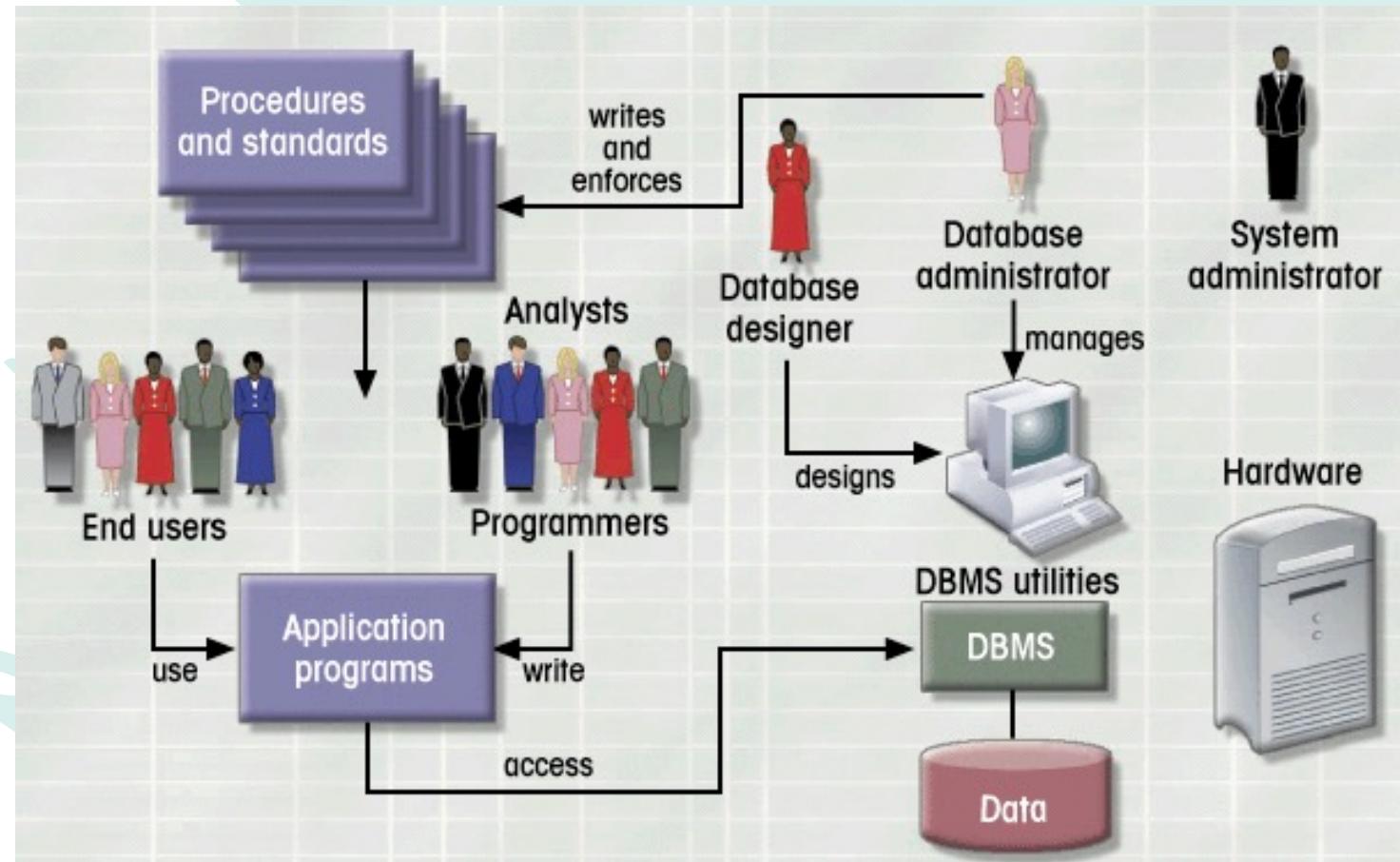
Database Systems

- Database consists of logically related data stored in a single repository
- Provides advantages over file system management approach
 - Eliminates inconsistency, data anomalies, data dependency, and structural dependency problems
 - Stores data structures, relationships, and access paths



Database System Environment

- Hardware
- Software
- People/Users
- Procedures → governance
- Data



KNOWLEDGE & SOFTWARE ENGINEERING

What Is a DBMS?

- A Database Management System (DBMS) is a software package designed to store and manage databases
- DBMS provides an environment that is both **convenient** and **efficient** to use.
 - Data independence and efficient access
 - Reduced application development time
 - Data integrity and security
 - Uniform data administration
 - Concurrent access, recovery from crashes

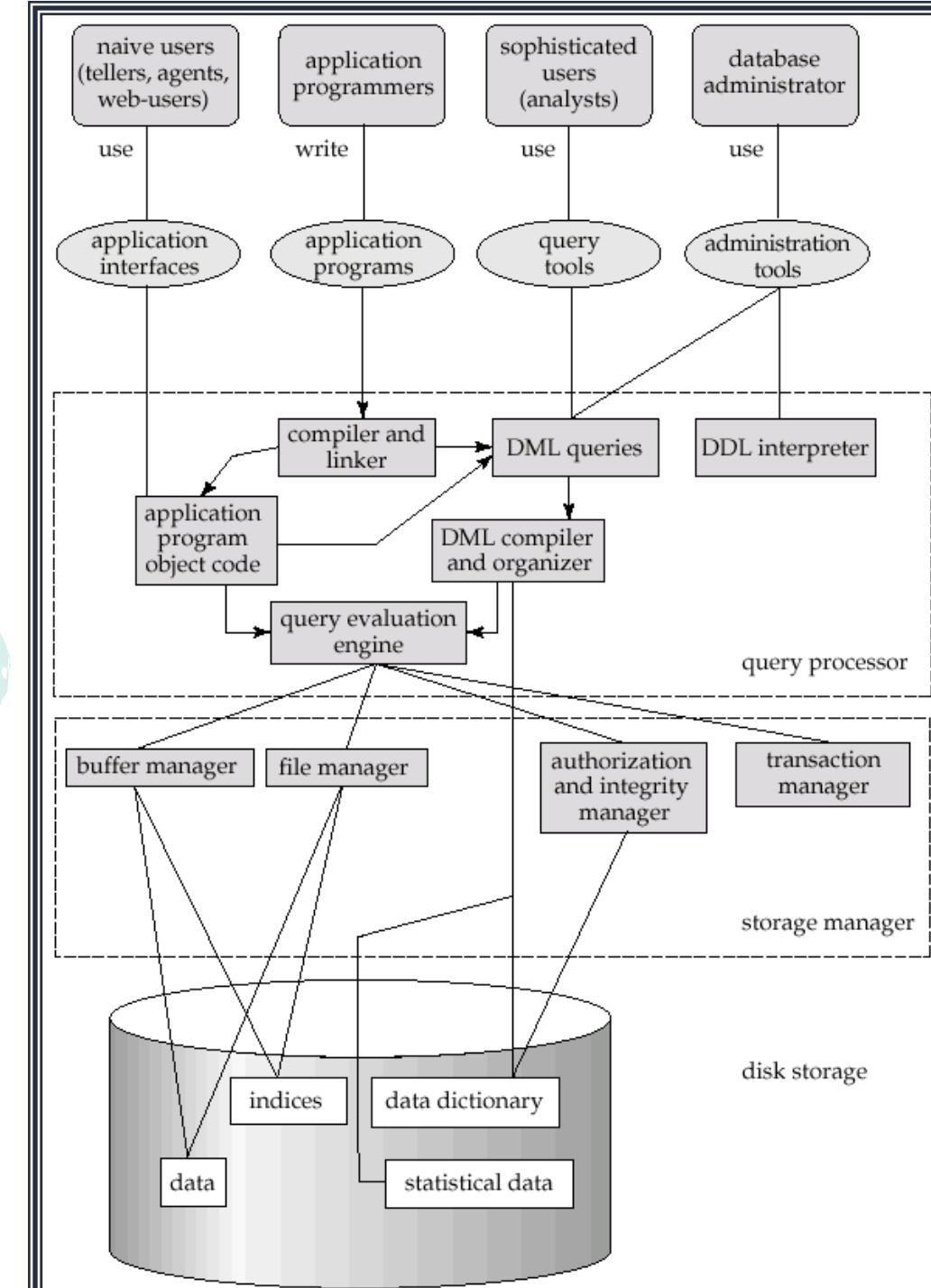


KNOWLEDGE & SOFTWARE ENGINEERING

Overall System Structure



KNOWLEDGE & SOFTWARE ENGINEERING



Database Engine

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The **functional components** of a database system can be divided into

The storage
manager

The query processor
component

The transaction
management
component



KNOWLEDGE & SOFTWARE ENGINEERING

Database Engine

The storage manager

The query processor
component

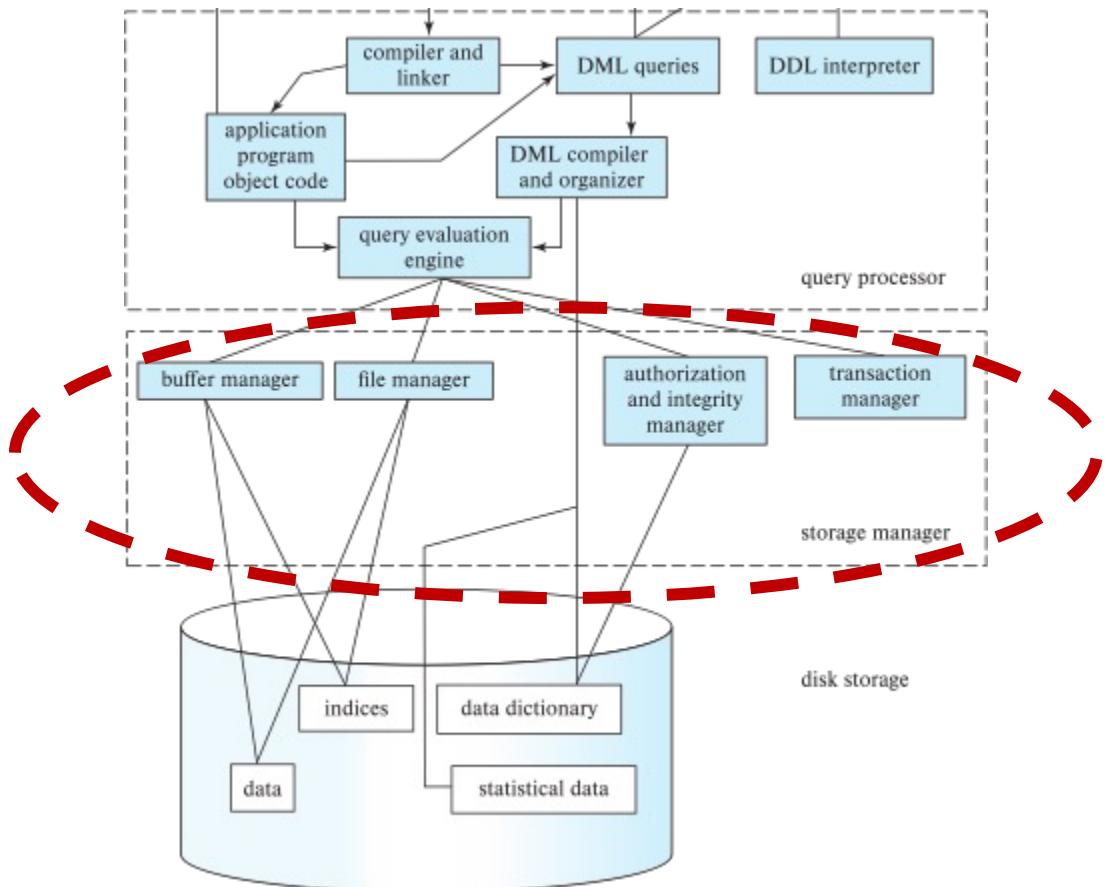
The transaction
management
component



KNOWLEDGE & SOFTWARE ENGINEERING

Storage Manager (1/2)

- A program module that provides the **interface between the low-level data stored in the database and the application programs** and queries submitted to the system.
- The storage manager is **responsible** to the following tasks:
 - Interaction with the OS file manager
 - Efficient storing, retrieving and updating of data
- The storage manager **components** include:
 - Authorization and integrity manager
 - Transaction manager
 - File manager
 - Buffer manager



Storage Manager (2/2)

The storage manager implements several data structures as part of the physical system implementation.

Data files

- store the database itself

Data dictionary

- stores metadata about the structure of the database, in particular the schema of the database.

Indices

- can provide fast access to data items. A database index provides pointers to those data items that hold a particular value.



Database Engine

The storage manager

The query processor
component

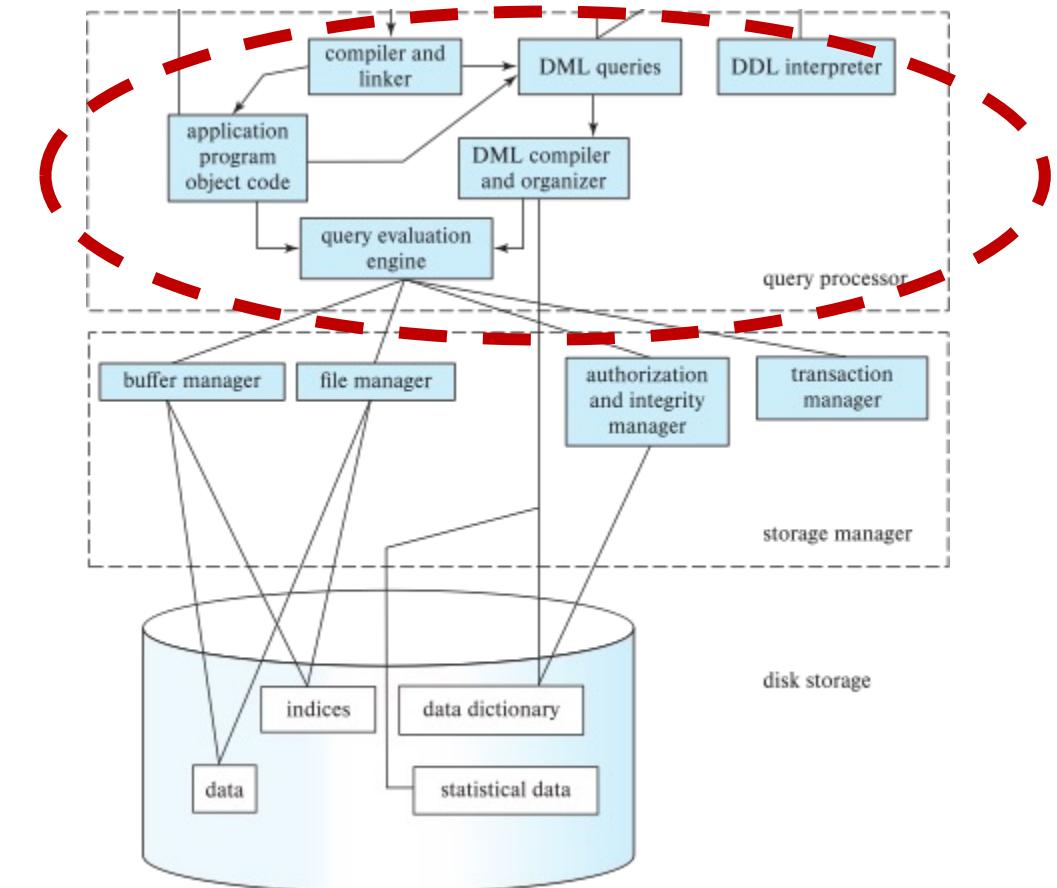
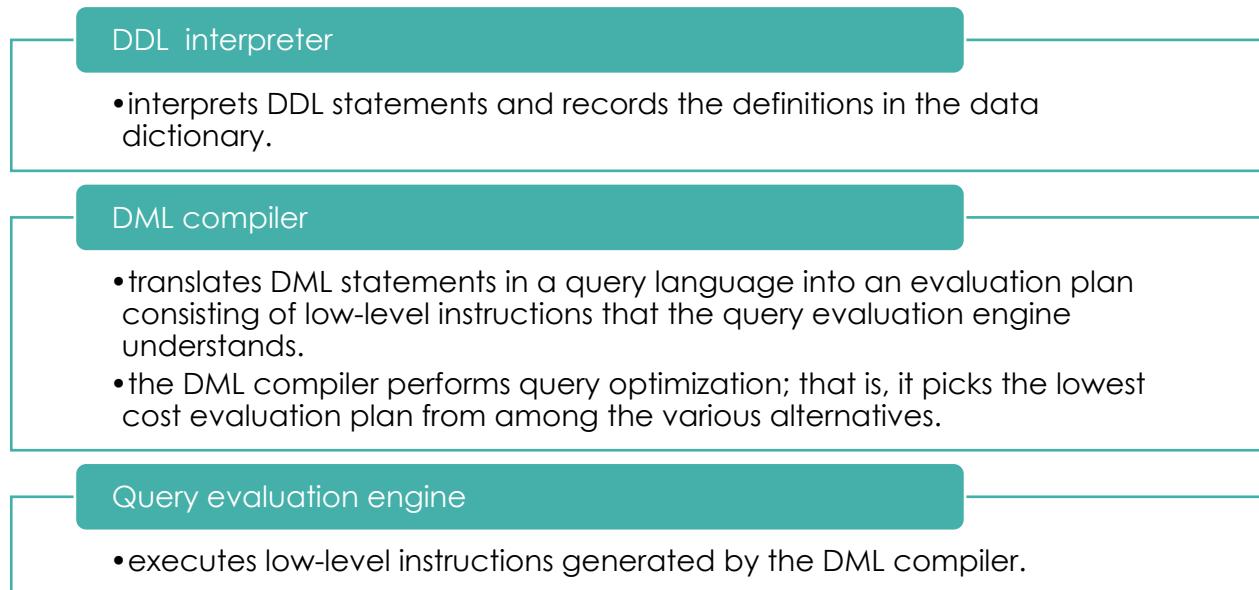
The transaction
management
component



KNOWLEDGE & SOFTWARE ENGINEERING

Query Processor (1/2)

- The query processor **components** include:

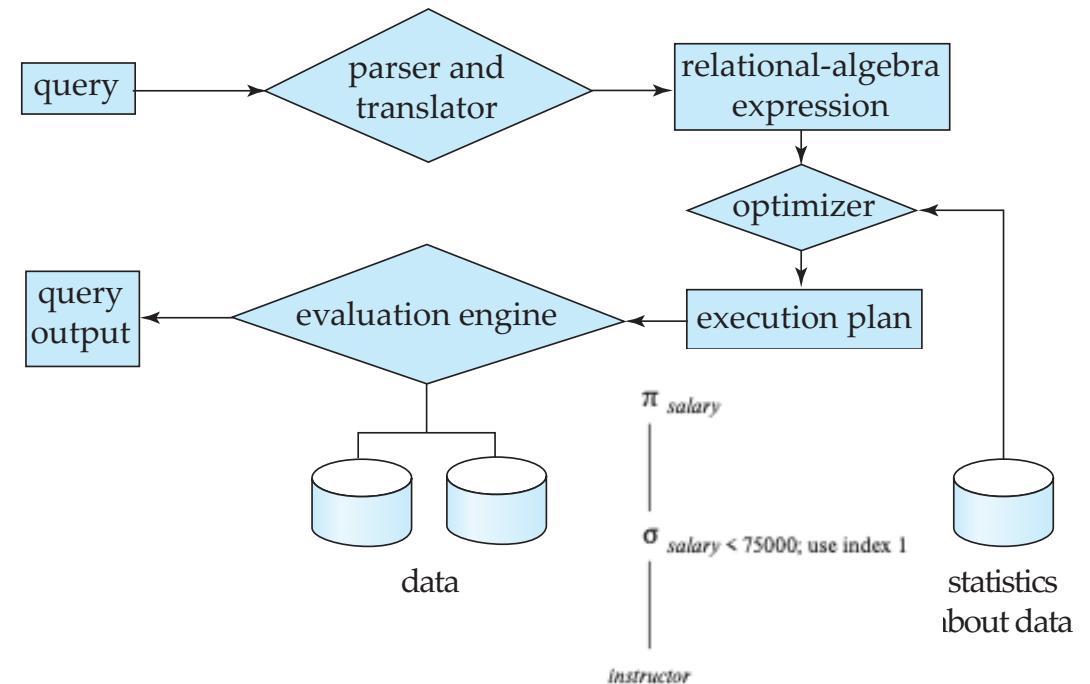


Query Processing (2/2)

1. Parsing and translation
2. Optimization
3. Evaluation

```
select salary
from instructor
where salary < 75000;
```

- $\sigma_{\text{salary} < 75000} (\Pi_{\text{salary}} (\text{instructor}))$
- $\Pi_{\text{salary}} (\sigma_{\text{salary} < 75000} (\text{instructor}))$



Database Engine

The storage manager

The query processor
component

The transaction
management
component



KNOWLEDGE & SOFTWARE ENGINEERING

Transaction Management

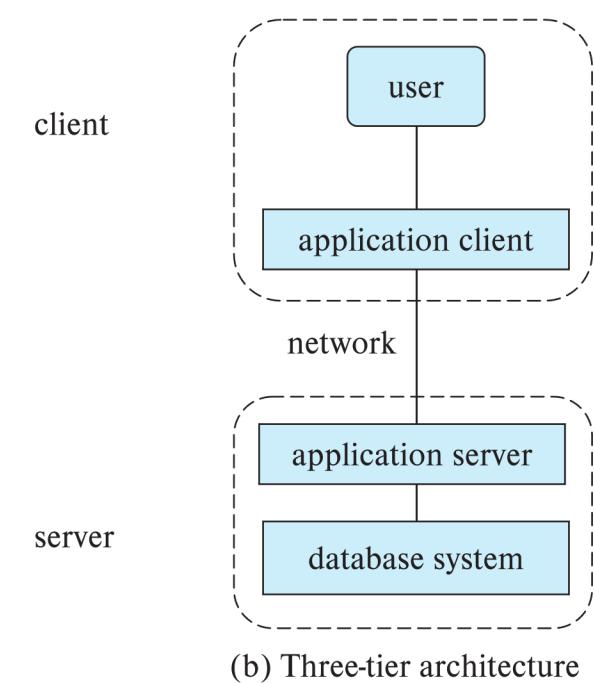
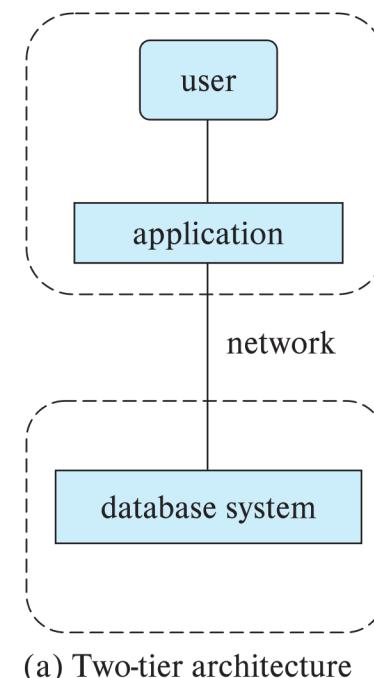
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database

Let T_i be transaction that transfers \$50 from account A to account B:

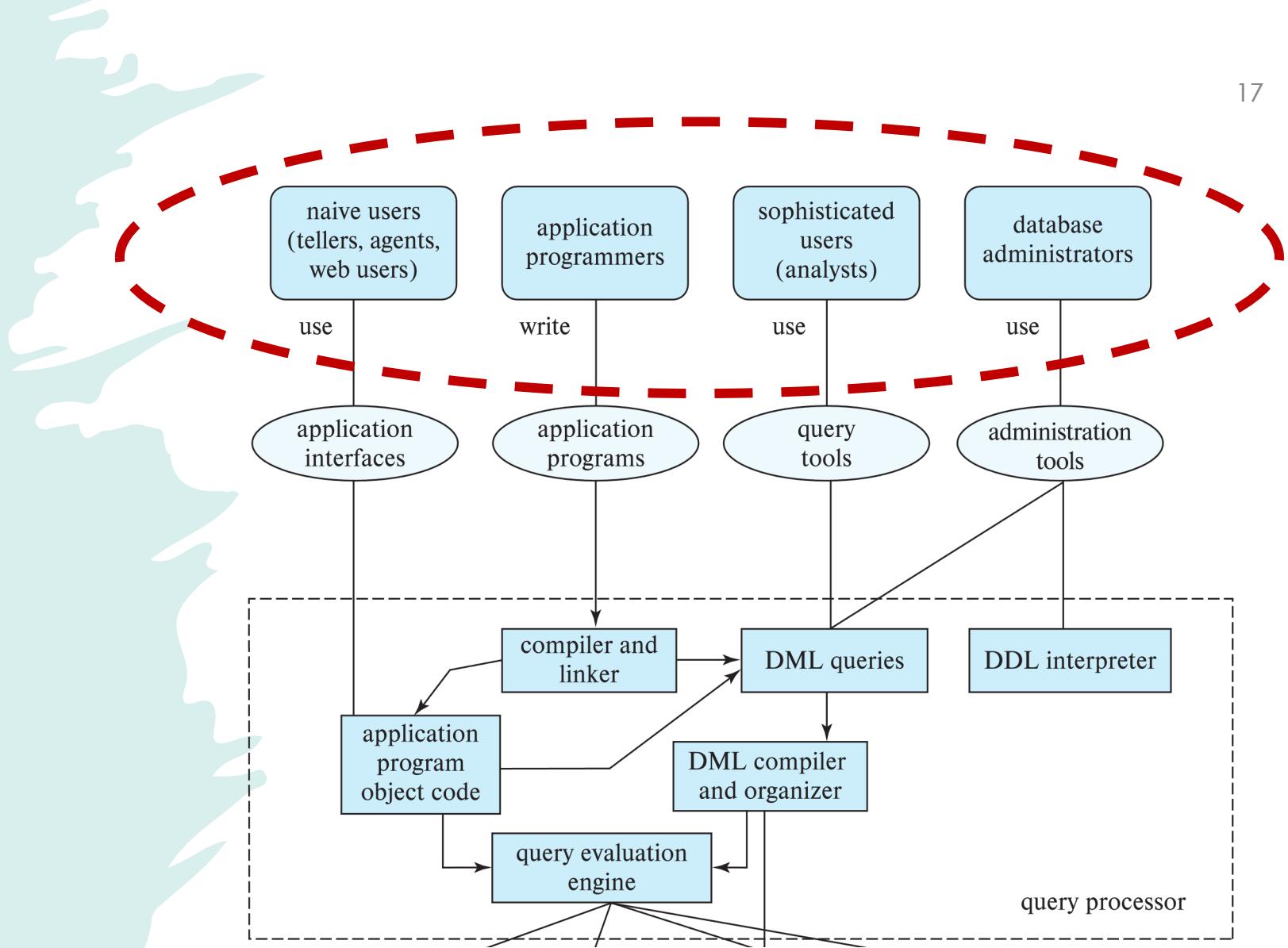
```
 $T_i$ : read(A);
       $A := A - 50;$ 
      write(A);
      read(B);
       $B := B + 50;$ 
      write(B).
```

Database Applications

- Two-tier architecture -- the application resides at the client machine, where it invokes database system functionality at the server machine
- Three-tier architecture -- the client machine acts as a front end and does not contain any direct database calls.
 - The client end communicates with an application server, usually through a forms interface.
 - The application server in turn communicates with a database system to access data.



Database Users



Database Administrator

- A person who has central control over the system is called a **database administrator (DBA)**.
- Functions of a DBA include:

Schema definition

Storage structure and access-method definition

Schema and physical-organization modification

Granting of authorization for data access

Routine maintenance

- Periodically backing up the database
- Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required
- Monitoring jobs running on the database



KNOWLEDGE & SOFTWARE ENGINEERING

IF3140 – Sistem Basis Data Performance Tuning

SEMESTER I TAHUN AJARAN 2024/2025



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from [Silberschatz's slides](#),
“Database System Concepts”, 7th ed.





Sumber

Silberschatz, Korth, Sudarshan:
“Database System Concepts”,
7th Edition

- **Chapter 25:** Performance Tuning & Performance Benchmark
- **Chapter 12.5:** RAID
- **Chapter 14:** Indexing



KNOWLEDGE & SOFTWARE ENGINEERING

Objectives



KNOWLEDGE & SOFTWARE ENGINEERING



Students are able to:

- Explain factors which impact database performance
- Describe several strategies to improve database performance
- Know database design and hardware aspects that respectively affect database transaction efficiency and system performance
- Know performance benchmarking

Outline



KNOWLEDGE & SOFTWARE ENGINEERING

Overview

Index +
Schema Tuning

Query Tuning

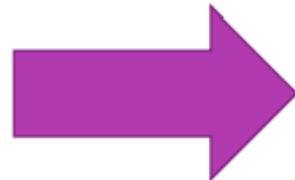
Transaction
Tuning

Hardware
Tuning

Performance
Benchmark

Overview

Database performance can be defined as the optimization of **resource** use to increase **throughput** and minimize **contention**, enabling the largest possible **workload** to be processed



Performance Tuning

- Adjusting various parameters and design choices to improve system performance (higher throughput)
- **General procedure:**
 - identifying bottlenecks, and
 - eliminating them.
- **Examples:**
 - Long delay in web application → further investigation → full relation scan query → adding an index
 - Long delay → further investigation → unnecessarily complicated query → rewrite the query



What affects performance?

- Hardware
 - CPU
 - Network
 - RAM
 - Disk
- Database Server parameter settings
- Database Design (schema)
- Indices on Database Tables
- SQL statement



KNOWLEDGE & SOFTWARE ENGINEERING

Bottlenecks Illustration

- Performance of most systems (at least before they are tuned) usually limited by performance of one or a few components:
 - **E.g.** 80% of the code may take up 20% of time and 20% of code takes up 80% of time
 - Worth spending most time on 20% of code that take 80% of time
- Bottlenecks may be in hardware (e.g. disks are very busy, CPU is idle), or in software
- Removing one bottleneck often exposes another
- De-bottlenecking consists of repeatedly finding bottlenecks, and removing them → heuristic



KNOWLEDGE & SOFTWARE ENGINEERING

Tuning Levels

- Can tune a database system at 3 levels:
 - **Higher level database design**,
E.g., Physical schema (indices, materialized view, horizontal splitting), queries, transactions, and logical schema
 - **Database system parameters**
E.g., Set buffer size to minimize disk I/O, set checkpointing intervals to limit log size.
 - **Hardware**
E.g., Add disks to speed up I/O, add memory to increase buffer hits, move to a faster processor.

Overview – File Organization

Database performance tuning is generally targeted to **minimize disk I/O** since it dominates the potential bottleneck

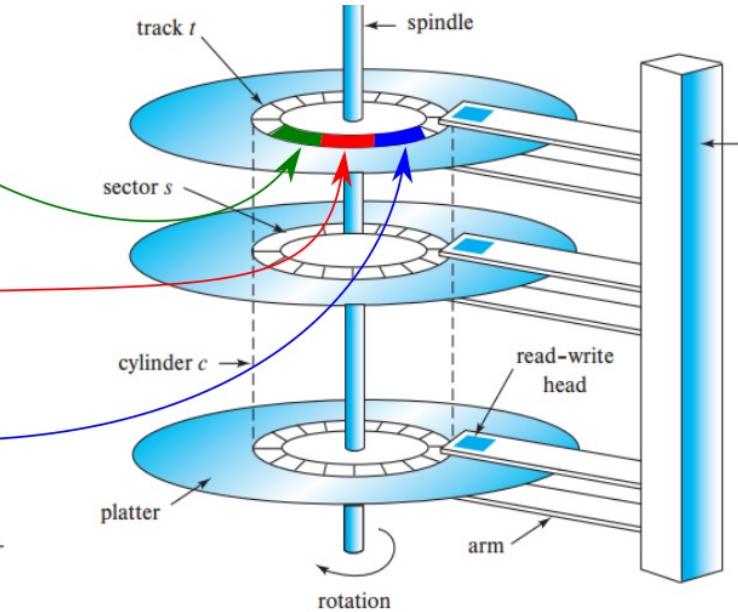
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Typical important constants

	HDD	SSD
Sector (HDD) /cell (SSD) size	512 B	2 B
Block / pages size	4 KB	4 KB
Random block read time (disk I/O)	10 ms Block access time: 5-20ms Block transfer time: 0.1ms	0.1 ms



KNOWLEDGE & SOFTWARE ENGINEERING



* Block access time in memory = ~100 ns (nano)

Overview

- After ER design, schema refinement, and the definition of views, we have the conceptual schema for our database.
- The next step is to: **(1) choose indices**, **(2) make clustering** decisions, and **(3) to refine the schemas** (if necessary), to meet performance goals.
- We must begin by understanding the **workload**:
 - The most important queries and how often they arise.
 - The most important updates and how often they arise.
 - The desired performance for these queries and updates.



KNOWLEDGE & SOFTWARE ENGINEERING

Understanding the Workload

- For each **query** in the workload:
 - Which **relations** does it **access**?
 - Which **attributes** are **retrieved**?
 - Which attributes are **involved in selection/join** conditions? How selective are these conditions likely to be?
- For each **update** in the workload:
 - Which attributes are involved in **selection/join conditions**? How selective are these conditions likely to be?
 - The type of update (INSERT/DELETE/UPDATE), and the **attributes** that are affected.

Decisions to Make

- What **indices** should we create?
 - Which **relations should have indices?** What **attribute(s)** should be the search key? Should we build **several indices**?
- For each index, what **kind of an index** should it be?
 - Clustered? Hash/tree? Dynamic/static? Dense/sparse?
- Should we **make changes** to the **logical schema**?
 - Consider alternative normalized schemas? (Remember, there are many choices in decomposing into BCNF, etc.)
 - Should we “undo” some decomposition steps and settle for a lower normal form? (*Denormalization*)
 - Horizontal partitioning, views, etc.



Indexing dan Index Tuning



KNOWLEDGE & SOFTWARE ENGINEERING

Index

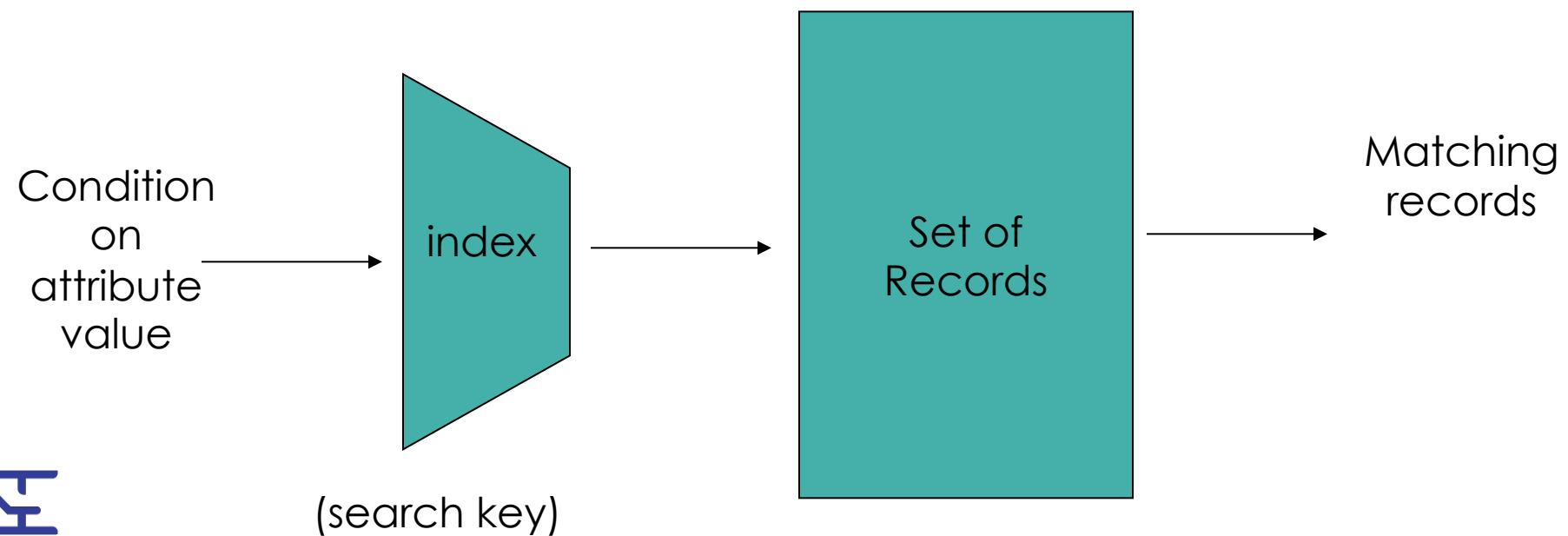
- Index → used to speed up access to desired data.
 - E.g., author catalog in library
- **Search Key** - attribute or set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form
 - search-key
 - pointer



KNOWLEDGE & SOFTWARE ENGINEERING

Index

- An index is a data structure that **supports efficient access** to data
- It facilitates searching, sorting, join operation, etc., efficiently instead of scanning all table rows
- Index files are typically **much smaller** than the original file





Schema Tuning



KNOWLEDGE & SOFTWARE ENGINEERING

Schema Tuning (I)

- Choice of logical schema should be guided by workload, in addition to redundancy issues:
 - We might consider **horizontal decompositions**.
 - We might consider **vertical decompositions**.
 - We may settle for a 3NF schema rather than BCNF.
 - Workload may influence choice we make in decomposing a relation into 3NF or BCNF.
 - We may further decompose a BCNF schema!
 - We might **denormalize** (i.e., undo a decomposition step), or we might add fields to a relation.
- If such changes are made after a database in use, called **schema evolution**; might mask changes by defining views

Schema Tuning (2)

- Can be done in several ways:

- **Splitting tables**

- Sometimes splitting normalized tables can improve performance
- Can split tables in two ways: (1) **Horizontally**; (2) **Vertically**
- Add complexity to the applications

- **Denormalization**

- Adding redundant columns
- Adding derived attributes
- Collapsing tables
- Duplicating tables



Query Tuning



KNOWLEDGE & SOFTWARE ENGINEERING

Tuning of Queries: Query Plan

- Optimizer **might not choose best plan**
 - Most DBMS support **EXPLAIN** command to see what plan is being used
- Makes decisions based on existing statistics
 - Statistics may be **old**
 - There is also **ANALYZE** command to re-compute the statistics
- Complex query containing **nested subqueries** are **not optimized** very well by many optimizers. The query can be **re-write using join** (decorrelation technique –see chapter 16.4.4 –)
- **Optimizer hints:** special instructions for the optimizer embedded in the SQL command text

Note: Rewriting query becomes less relevant since optimizers goes better

**TABLE
11.5**

Optimizer Hints

HINT	USAGE
ALL_ROWS	<p>Instructs the optimizer to minimize the overall execution time—that is, to minimize the time needed to return all rows in the query result set. This hint is generally used for batch mode processes. For example:</p> <pre>SELECT /*+ ALL_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;</pre>
FIRST_ROWS	<p>Instructs the optimizer to minimize the time needed to process the first set of rows—that is, to minimize the time needed to return only the first set of rows in the query result set. This hint is generally used for interactive mode processes. For example:</p> <pre>SELECT /*+ FIRST_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;</pre>
INDEX(name)	<p>Forces the optimizer to use the P_QOH_NDX index to process this query. For example:</p> <pre>SELECT /*+ INDEX(P_QOH_NDX) */ * FROM PRODUCT WHERE P_QOH < 10;</pre>



Tuning of Queries: Set Orientation

- In an application program, it is often that a query is executed frequently, but with different values for a parameter.
- **Set orientation** → performing fewer calls to database

```
select sum(salary)  
from instructor  
where dept_name= ?
```

Instead of this



```
select dept_name, sum(salary)  
from instructor  
group by dept_name;
```

use this



KNOWLEDGE & SOFTWARE ENGINEERING



Transaction Tuning



KNOWLEDGE & SOFTWARE ENGINEERING

Tuning of Transactions

- Long transactions (typically **read-only**) that examine large parts of a relation **result in lock contention** with update transactions
 - E.g. large query to compute bank statistics and regular bank transactions
- To reduce contention
 - Use multi-version concurrency control
 - E.g. Oracle “**snapshots**” which support multi-version 2PL
 - Use degree-two consistency (cursor-stability) for long transactions
 - **Drawback:** result may be approximate

without lock

Tuning of Transactions (Cont.)

- Long update transactions cause several problems
 - Exhaust lock space
 - Exhaust log space
 - and also greatly increase recovery time after a crash, and may even exhaust log space during recovery if recovery algorithm is badly designed!
- Use **mini-batch** transactions to limit number of updates that a single transaction can carry out.
 - Split large transaction into batch of “mini-transactions”, each performing part of the updates
 - In case of failure during a mini-batch, must complete its remaining portion on recovery, to ensure atomicity.



Hardware Tuning



KNOWLEDGE & SOFTWARE ENGINEERING

Tuning of Hardware

- Even well-tuned transactions typically require a few I/O operations
 - Typical disk supports about **100 random I/O** operations per second
 - Suppose each transaction requires just **2 random I/O operations**. Then to support **n transactions per second**, we need to **stripe data** across **$n/50$ disks** (ignoring skew)
- Number of I/O operations per transaction can **be reduced** by **keeping more data in memory**
 - If all data is in memory, I/O needed only for writes
 - Keeping frequently used data in memory reduces disk accesses, reducing number of disks required, but has a memory cost

Hardware Tuning: Which Data to Keep in Memory?

- Economic consideration is popularly used to answer question above, which is **finding the break even point** between **disk access cost** and **memory cost**.
- Disk access cost
 - If a block is accessed once in ***m*** second, then:

$$\text{disk access cost} = \frac{\text{price per disk}}{\text{number of access per second} \times m}$$

- Memory cost by keeping a block of data

$$\text{memory cost} = \frac{\text{price per MB of memory}}{\text{number of blocks per MB}}$$



KNOWLEDGE & SOFTWARE ENGINEERING

Hardware Tuning: Which Data to Keep in Memory?

- In 1987, price per disk = \$30,000 with 15 random access/s, while price per 1MB memory = \$5,000 with block size = 1KB.
- Suppose we have blocks of data that are **accessed once per 2 seconds**, then:

- **disk access cost** = $\frac{\text{price per disk}}{\text{number of access per second} \times m} = \frac{\$30,000}{15 \text{ access/s} \times 2\text{s}} = \$1,000 \text{ per block access}$

- **memory cost** = $\frac{\text{price per MB of memory}}{\text{number of blocks per MB}} = \frac{\$5,000}{1,000 \text{ blocks}} = \5 per block

- Buffering this data:
 - Needs \$5 memory cost, reduces \$1,000 disk access cost → **WORTH!**
- To find the break even point of **m**, thus:

$$m = \frac{\text{price per disk}}{\text{price per MB memory}} \times \frac{\text{numb. blocks per MB}}{\text{numb. disk access}} = \frac{30,000}{5,000} \times \frac{1,000}{15} = 400 \text{ s} \approx 5 \text{ mins.}$$

Thus, in that time, data accessed every 400s or less is worth to buffer in the memory → buy more memory!



Hardware Tuning: Five-Minute Rule

- Using formula and data table below, we can get m from time to time:

$$m = \frac{\text{price per disk}}{\text{price per MB memory}} \times \frac{\text{numb. blocks per MB}}{\text{numb. disk access}}$$

Metric	DRAM				HDD				SATA Flash SSD	
	1987	1997	2007	2017	1987	1997	2007	2017	2007	2017
Unit price(\$)	5k	15k	48	80	30k	2k	80	49	1k	560
Unit capacity	1MB	1GB	1GB	16GB	180MB	9GB	250GB	2TB	32GB	800GB
\$/MB	5k	14.6	0.05	0.005	83.33	0.22	0.0003	0.00002	0.03	0.0007
Random IOPS	-	-	-	-	15	64	83	200	6.2k	67k (r)/20k (w)
Sequential b/w (MB/s)	-	-	-	-	1	10	300	200	66	500 (r)/460 (w)

Source: Appuswamy, et. all. "The five minute rule thirty years later...." 2017.

In 1987, $m = 400\text{s} \approx 5 \text{ minutes}$
→ **five minute rule!**

Tier	1987	1997	2007	2017
DRAM-HDD	5m	5m	1.5h	4h
DRAM-SSD	-	-	15m	7m (r) / 24m (w)

Hardware Tuning: One-Minute Rule

- Prices of disk and memory have changed greatly over the years, but the ratios have not changed much from 1987 to 1997, so during this period:
 - Rules remain as **5 minute**, not 1 hour or 1 second rules (for random access)
 - But rules change after that period of time, e.g., in 2007 and 2017 which are 1.5 hours and **4 hours** rules!
- **For sequentially accessed data**, more blocks can be read per second. Assuming sequential reads of 1MB of data at a time:
1-minute rule: sequentially accessed data that is accessed once or more in a minute should be kept in memory



KNOWLEDGE & SOFTWARE ENGINEERING

Hardware Tuning: RAID

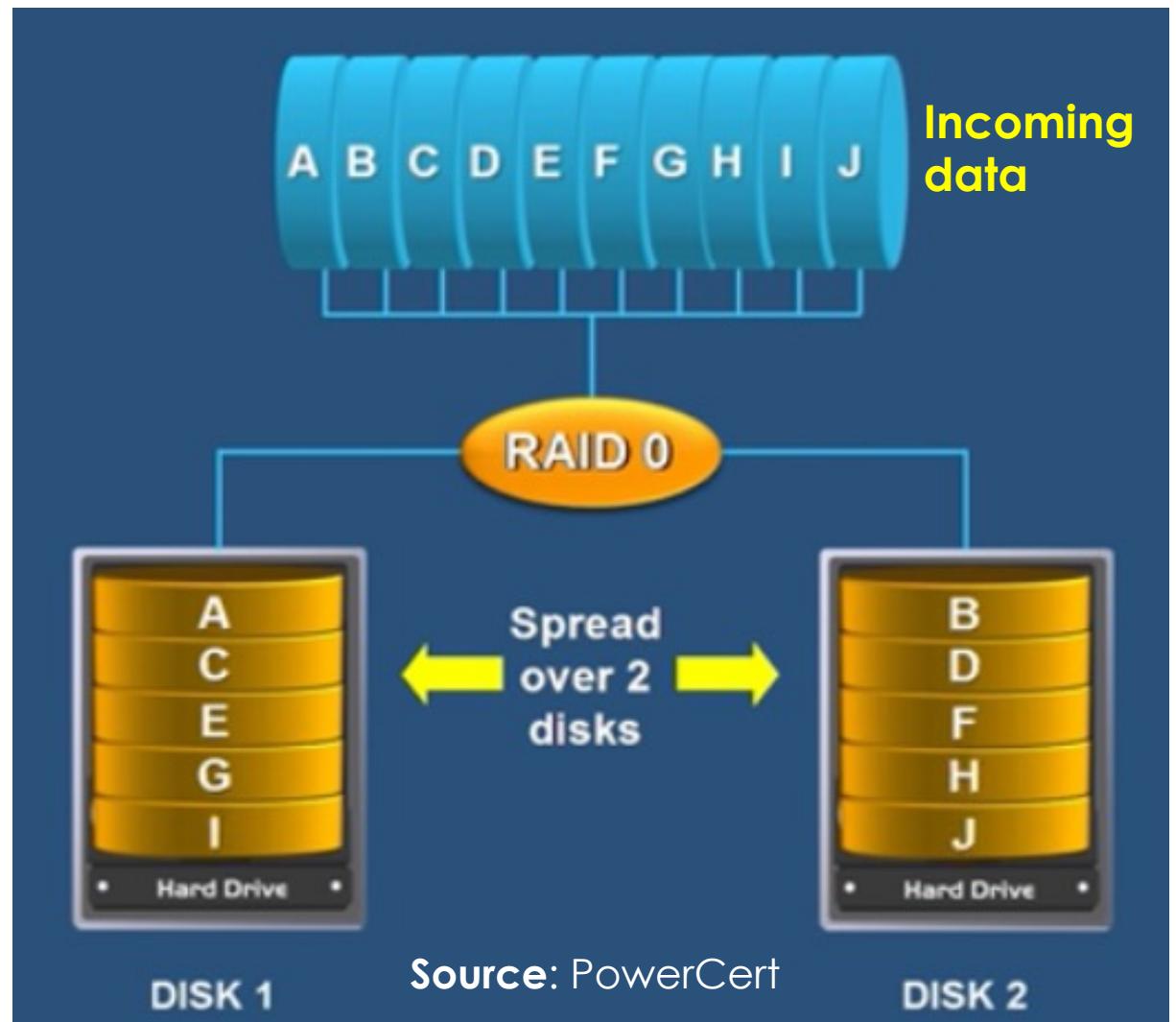
- Redundant Array of Independent Disc (RAID):
 - Disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
 - **high capacity** and **high speed** by using multiple disks in parallel,
 - **high reliability** by storing data redundantly, so that data can be recovered even if a disk fails
 - **Stripping** – strip data across multiple disk to increase transfer rate (bit-level stripping and block level stripping)
 - **Redundancy** – store extra information that can be used to rebuild information lost in a disk failure



KNOWLEDGE & SOFTWARE ENGINEERING

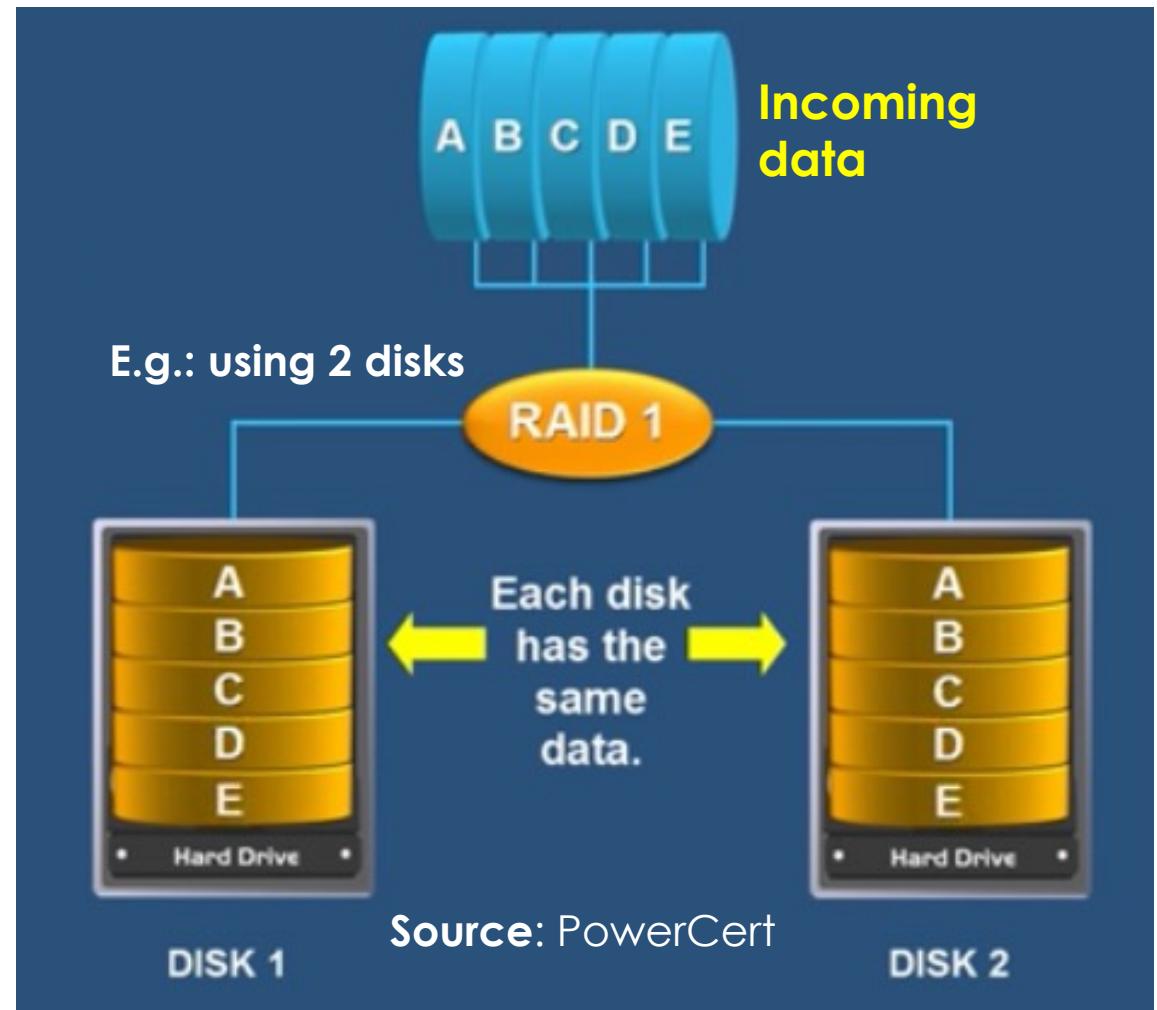
Hardware Tuning: RAID 0

- Benefit:
 - Strip the data across multiple disk → **fast read**
- Drawback:
 - **No fault tolerance** → if a disk fails, the data is lost



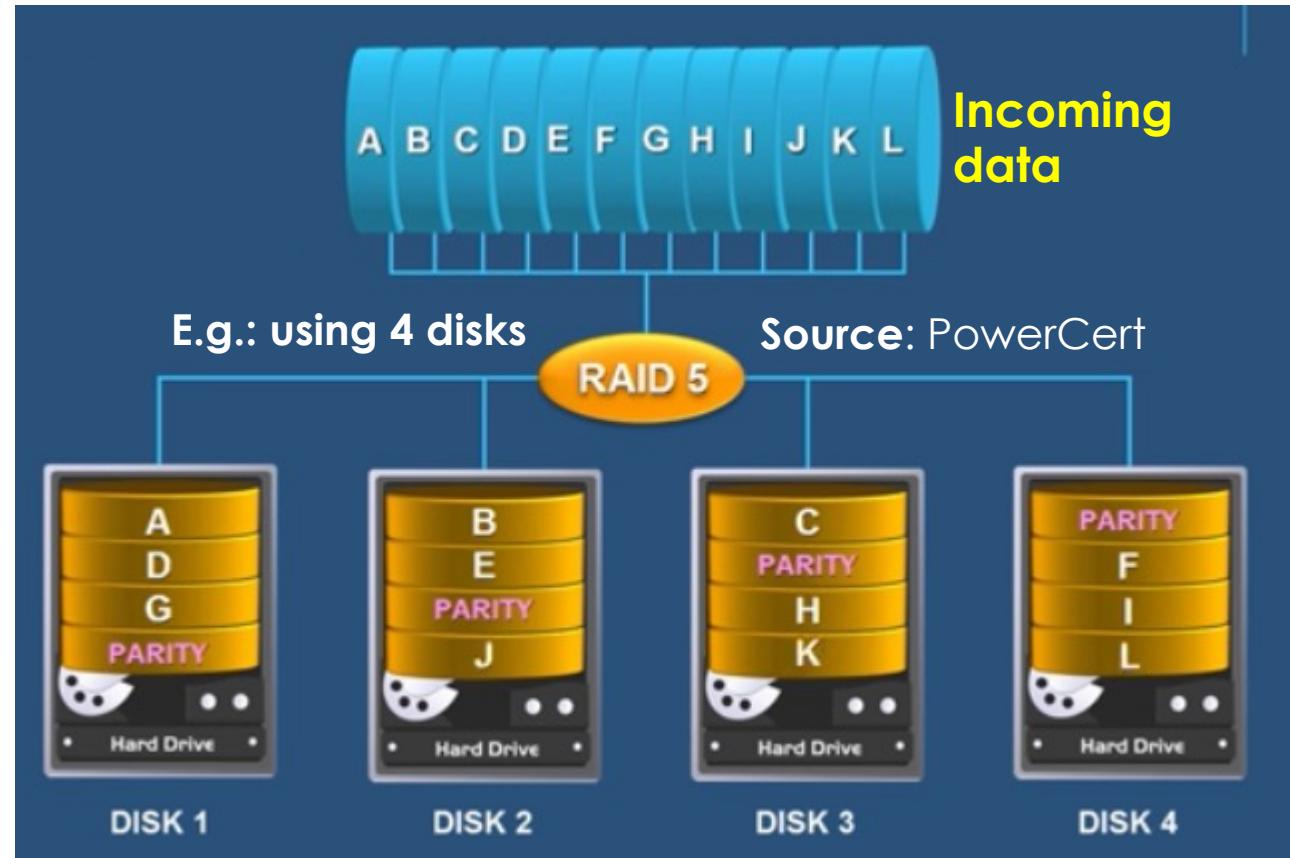
Hardware Tuning: RAID 1

- Benefit:
 - Copy (mirror) the data across multiple disk → **fault tolerant**
 - Relatively easy to rebuild upon the failure
- Drawback:
 - **Need more disk** to store the data
 - **E.g.**, for 1 data copy, only 50% disk for the actual data



Hardware Tuning: RAID 5

- Benefit:
 - Store 1 parity data → **fault tolerant** up to 1 failure
- Drawback:
 - **Need more disk** to store the data
 - **E.g.**, for 4 disks setup, only 75% disk for the actual data
 - More cost to write, and rebuild upon failure
 - **Raid 6**: Similar to raid 5, but store more redundancies



E.g.: using even parity $D_1, D_2, D_3, D_4 = 0, 1, 1, 0$. Any disk failure, we still can recover.

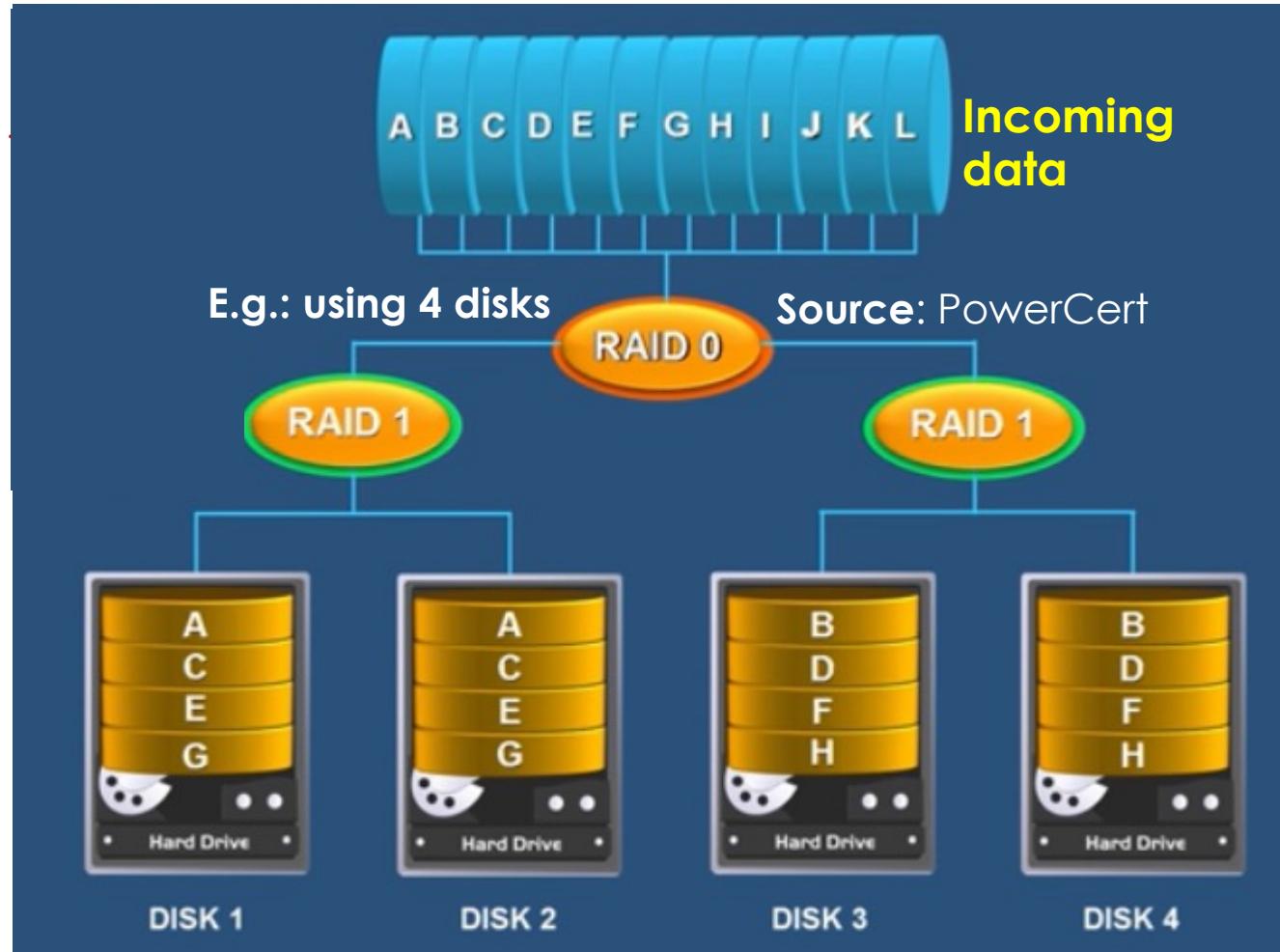
Hardware Tuning: RAID 10

- Benefit:
 - Combine raid 0 (stripping) and raid 1 (redundancy)
→ **fast read and fault tolerance**
- Drawback:
 - **Need more disk** (at least 4) to store the data
 - **E.g.**, for 4 disks setup, only 50% disk for the actual data

Note: Raid 2, 3, 4 are not used in practice anymore.



KNOWLEDGE & SOFTWARE ENGINEERING



Hardware Tuning: Choice of RAID Level

- To use RAID 1 or RAID 5?
 - Depends on ratio of reads and writes
 - RAID 5 requires 2 block reads and 2 block writes to write out one data block, meanwhile RAID 1 only needs 2 block writes
- If an application requires **r reads** and **w writes** per second
 - **RAID 1** requires **$r + 2w$ I/O** operations per second
 - **RAID 5** requires **$r + 4w$ I/O** operations per second
- For reasonably large r and w, this requires lots of disks to handle workload
 - RAID 5 may require more disks than RAID 1 to handle load!
 - Apparent saving of number of disks by RAID 5 (by using parity, as opposed to the mirroring done by RAID 1) may be illusory!
- **Rule of Thumb:** RAID 5 is fine when writes are rare and data is very large, but RAID 1 is preferable otherwise

Performance Simulation

- Performance simulation using queuing model useful to predict bottlenecks as well as the effects of tuning changes, even without access to real system
- Simulation model is quite detailed, but usually omits some low level details
 - Model **service time**, but disregard details of service
 - **E.g.** approximate disk read time by using an average disk read time
- Experiments can be run on model, and provide an estimate of measures such as average throughput/response time
- Parameters can be tuned in model and then replicated in real system
 - **E.g.** number of disks, memory, algorithms, etc.



Performance **Benchmark**



KNOWLEDGE & SOFTWARE ENGINEERING

Performance Benchmarks

- Suites of tasks used to quantify the performance of software systems
- Important in comparing database systems, especially as systems become more standards compliant.
- Commonly used performance measures:
 - Throughput (transactions per second, or tps)
 - Response time (delay from submission of transaction to return of result)
 - Availability or mean time to failure



KNOWLEDGE & SOFTWARE ENGINEERING

Performance Benchmarks (Cont.)

- Beware when computing average throughput of different transaction types
 - E.g., suppose a system runs transaction type **A at 99 tps** and transaction **type B at 1 tps**.
 - Given an equal mixture of types A and B, throughput **is not $(99+1)/2 = 50 \text{ tps}$** .
 - Running one transaction of each type takes time $0.01+1=1.01$ seconds, giving a throughput of **1.98 tps**.
 - To compute average throughput of **n** transactions with tps of t_i , use **harmonic mean**:

$$\frac{n}{\frac{1}{t_1} + \frac{1}{t_2} + \dots + \frac{1}{t_n}}$$

- **Interference** (e.g. lock contention) makes even this incorrect if different transaction types run concurrently

Database Application Classes

- **Online transaction processing (OLTP)**
 - requires high concurrency and clever techniques to speed up commit processing, to support a high rate of update transactions.
- **Decision support applications**
 - including **online analytical processing**, or **OLAP** applications
 - require good query evaluation algorithms and query optimization.
- Architecture of some database systems tuned to one of the two classes
 - E.g. Teradata is tuned to decision support
- Others try to balance the two requirements
 - E.g. Oracle, with snapshot support for long read-only transaction

Benchmarks Suites

- The **Transaction Processing Council (TPC)** benchmark suites are widely used.
 - **TPC-A** and **TPC-B**: simple **OLTP application** modeling a bank teller application with and without communication
 - Not used anymore
 - **TPC-C**: complex OLTP application modeling an inventory system
 - Current standard for OLTP benchmarking



KNOWLEDGE & SOFTWARE ENGINEERING

Benchmarks Suites (Cont.)

- TPC benchmarks (cont.)
 - **TPC-D:** complex **decision support application** with 17 queries
 - Refined by TPC-H and TPC-R
 - **TPC-H:** (H for ad hoc) based on TPC-D with some extra queries
 - Models ad hoc queries which are not known beforehand
 - Total of 22 queries with emphasis on aggregation
 - prohibits materialized views
 - permits indices only on primary and foreign keys
 - **TPC-R:** (R for reporting) same as TPC-H, but without any restrictions on materialized views and indices
 - **TPC-W:** (W for Web) End-to-end Web service benchmark modeling a Web bookstore, with combination of static and dynamically generated pages

TPC Performance Measures

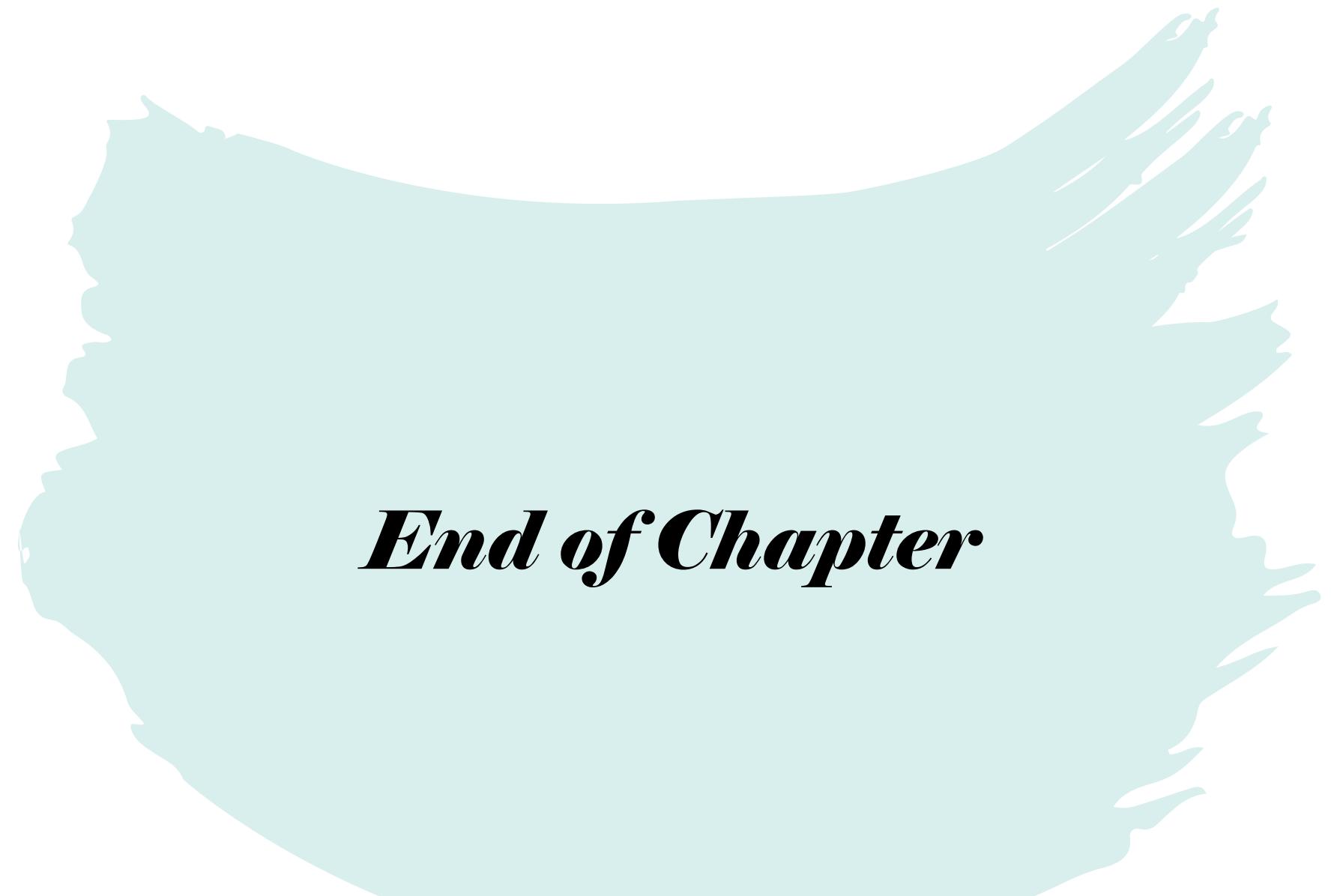
- TPC performance measures
 - **transactions-per-second** with specified constraints on response time
 - **transactions-per-second-per-dollar** accounts for cost of owning system
- TPC benchmark requires database sizes to be scaled up with increasing transactions-per-second
 - reflects real world applications where more customers means more database size and more transactions-per-second
- External audit is mandatory to claim TPC performance numbers
 - TPC performance claims can be trusted



KNOWLEDGE & SOFTWARE ENGINEERING

TPC Performance Measures

- Two types of tests for TPC-H and TPC-R
 - **Power test:** runs queries and updates sequentially, then takes mean to find queries per hour
 - **Throughput test:** runs queries and updates concurrently
 - multiple streams running in parallel each generates 22 queries, with one parallel update stream
 - **Composite query per hour metric:** square root of product of power and throughput metrics
 - **Composite price/performance metric:** dividing the system price by the metric



End of Chapter



KNOWLEDGE & SOFTWARE ENGINEERING

IF3140 – Sistem Basis Data Storage and File Structure

SEMESTER II TAHUN AJARAN 2024/2025



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from [Silberschatz's slides](#),
“Database System Concepts”, 7th ed.



Modified from Silberschatz's slides,
“Database System Concepts”, 7th ed.

Sumber

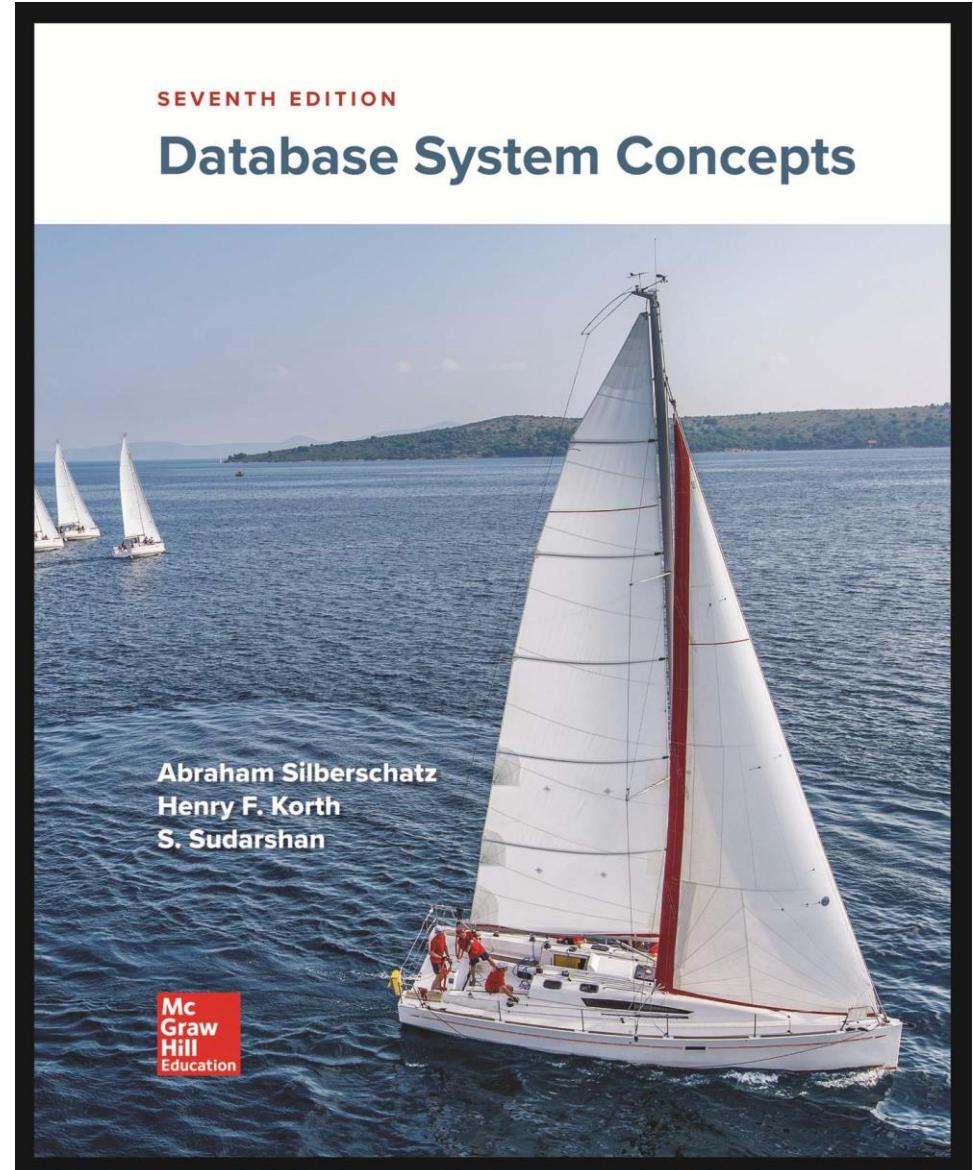
Silberschatz, Korth, Sudarshan:
“Database System
Concepts”, 7th Edition

- **Chapter 12:** Physical Storage Systems
- **Chapter 13:** Data Storage Structures



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
“Database System Concepts”, 7th ed.



Objectives



KNOWLEDGE & SOFTWARE ENGINEERING



Mahasiswa mampu:

- menjelaskan media penyimpanan data secara fisik
- menjelaskan organisasi file, bagaimana *record* disimpan dalam *file*,
- Menjelaskan tantangan terhadap kinerja basis data berkaitan dengan bagaimana data disimpan.



Outline

Classification of Physical Storage Media

Performance Measures of Disks

Disk Block Access

File Organization

Organization of Records in Files

Data Dictionary Storage



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Physical Storage Systems



KNOWLEDGE & SOFTWARE ENGINEERING

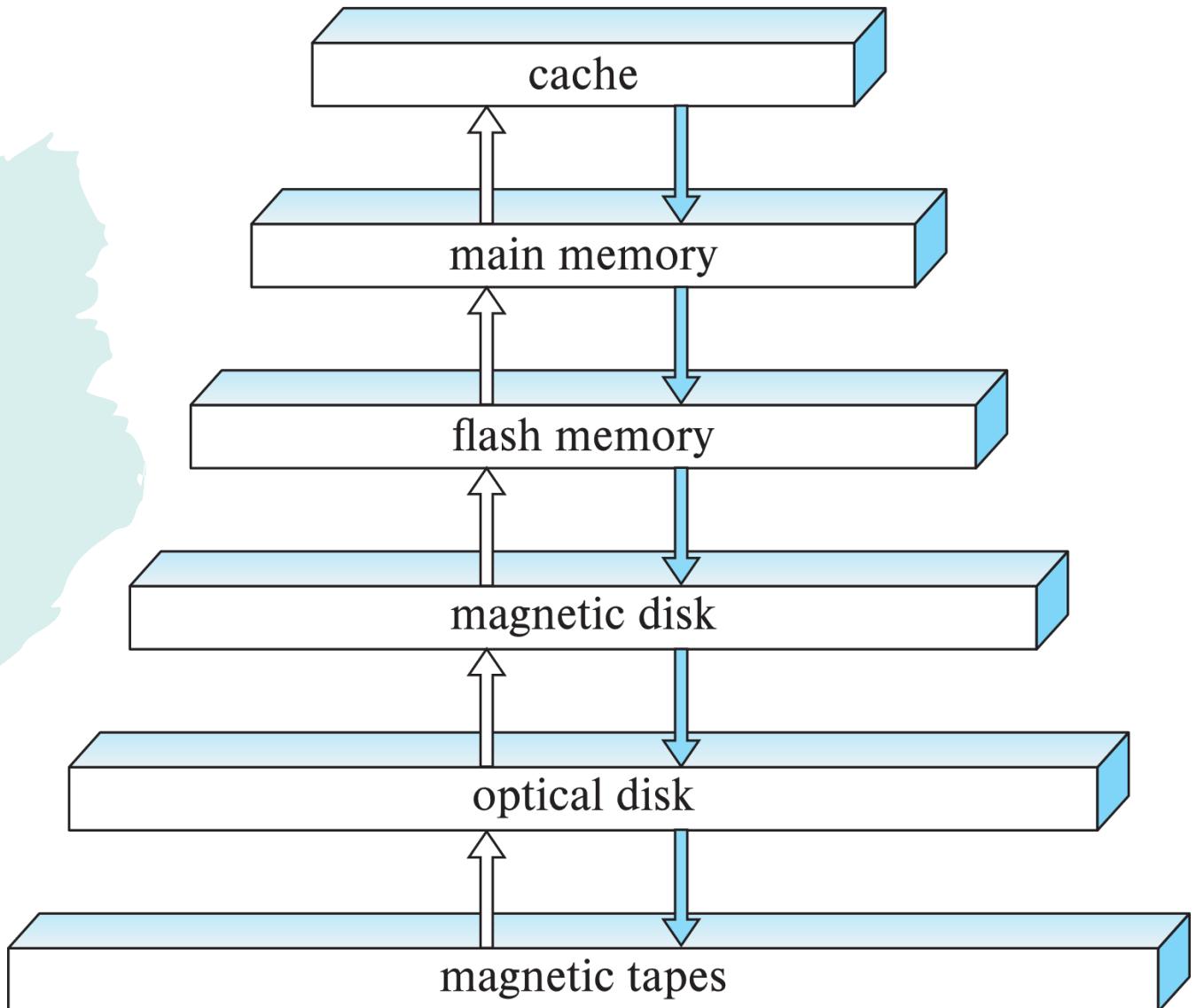


Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Classification of Physical Storage Media

- Different types of storage:
 - **volatile storage**: loses contents when power is switched off (ex: RAM)
 - **non-volatile storage**: (ex: SSD)
 - Contents persist even when power is switched off.
 - Includes secondary and tertiary storage, as well as battery-backed up main-memory.
- Factors affecting choice of storage media include
 - Speed with which data can be accessed
 - Cost per unit of data
 - Reliability

Storage Hierarchy



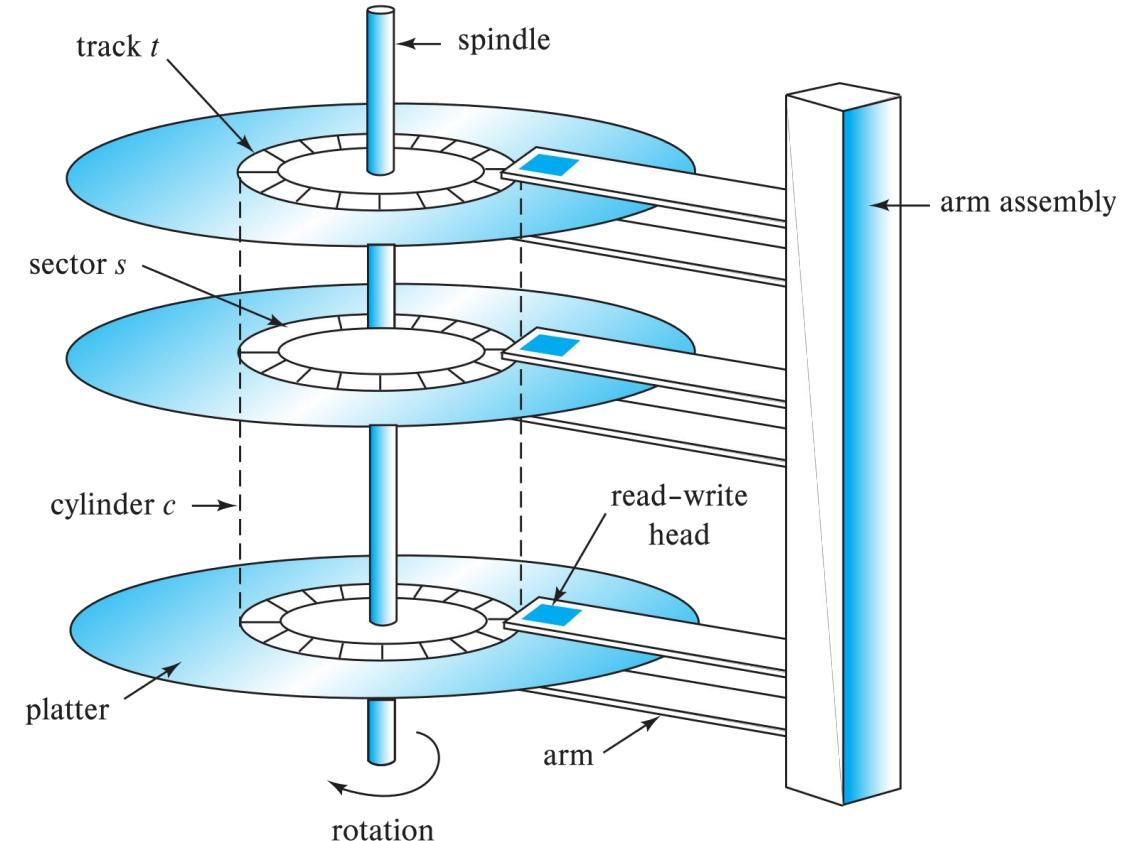
Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.



KNOWLEDGE & SOFTWARE ENGINEERING

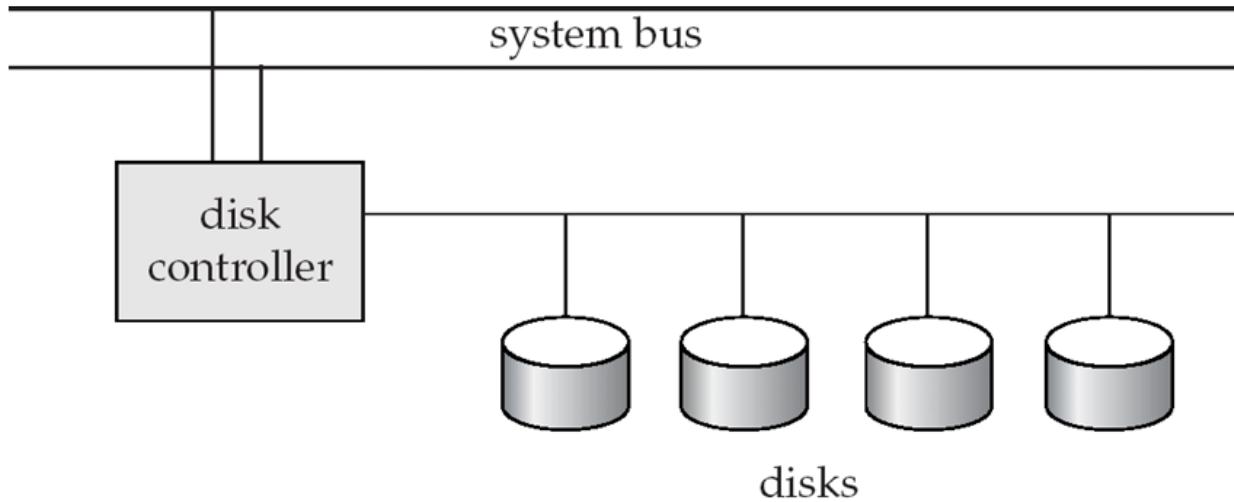
Magnetic Hard Disk Mechanism

- Surface of platter divided into circular **tracks**
- Each track is divided into **sectors**
 - A sector is the smallest unit of data that can be read or written
- Read-Write Head
- Head-disk assemblies
- **Cylinder i** consists of i -th track of all the platters



Schematic diagram of magnetic disk drive

Disk Subsystem



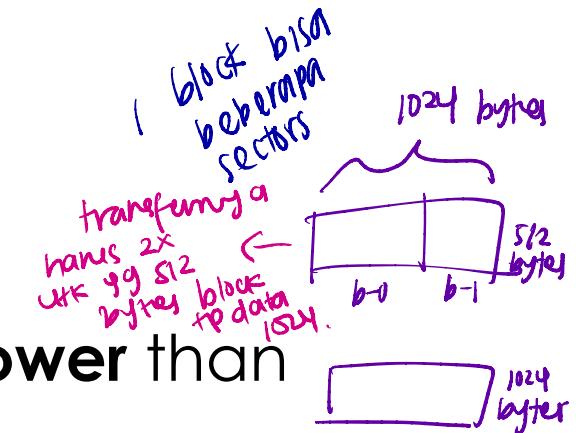
- **Disk controller** – interfaces between the computer system and the disk drive hardware.
 - Accepts high-level commands to read or write a sector
 - Initiates actions such as moving the disk arm to the right track and actually reading or writing the data
 - Computes and attaches **checksums** to each sector to verify that data is read back correctly
 - Performs **remapping of bad sectors**
- Multiple disks connected to a computer system through a disk controller

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins, consists of:
 - **Seek time** – time it takes to reposition the arm over the correct track
 - **Rotational latency** – time it takes for the sector to be accessed to appear under the head
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk
- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
 - Typically 3 to 5 years
 - MTTF decreases as disk ages

Disk Block Access

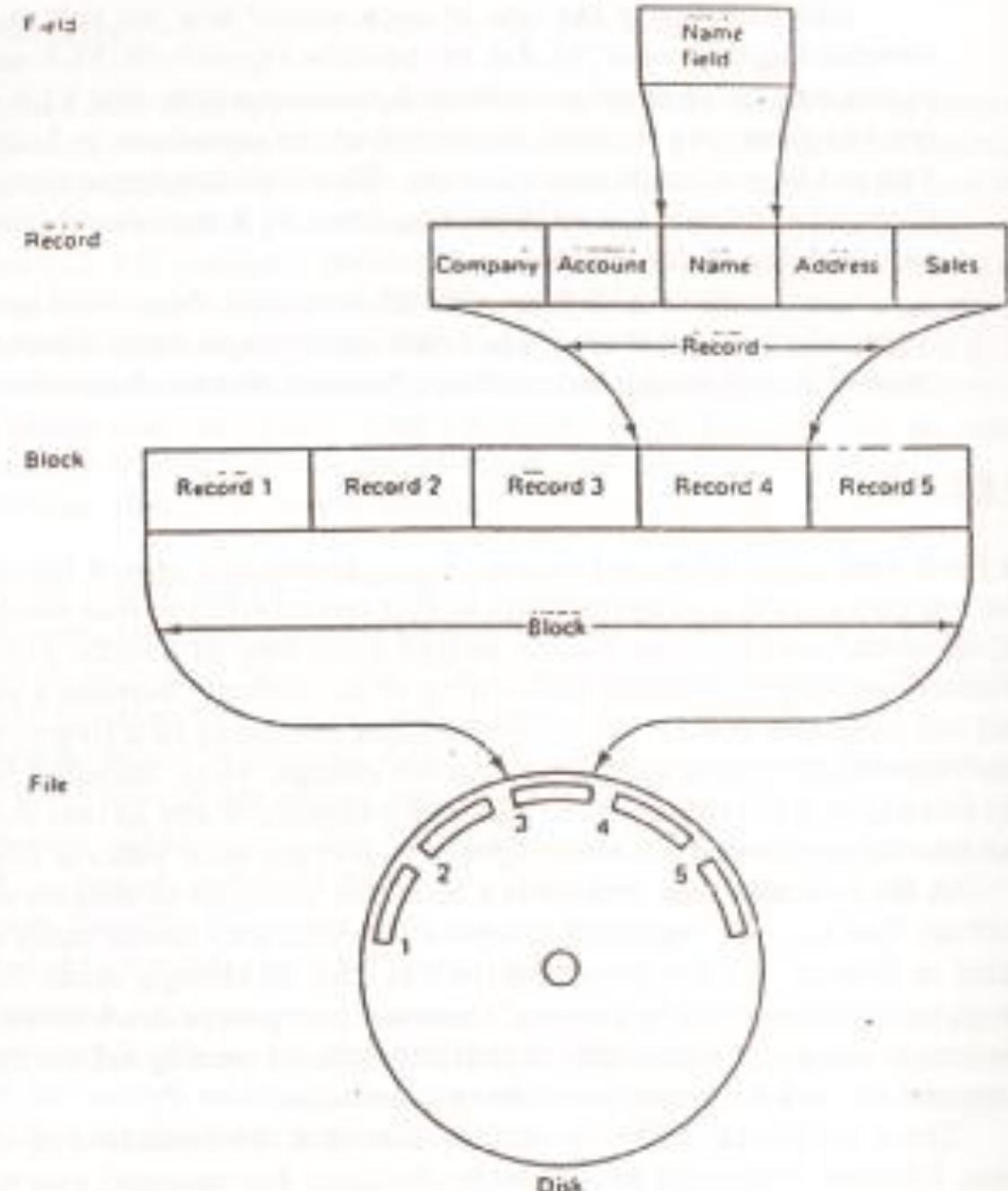
- A database file is partitioned into fixed-length storage units called **blocks**. Blocks are units of both storage allocation and data transfer.
- **Block** – a contiguous sequence of sectors from a single track
 - data is transferred between disk and main memory in blocks
 - sizes range from 512 bytes to several kilobytes
 - Smaller blocks: more transfers from disk
 - Larger blocks: more space wasted due to partially filled blocks
 - Typical block sizes today range from 4 to 16 kilobytes
- Access to data on disk is several orders of magnitude **slower** than access to data in main memory
- Database system seeks to minimize the number of block transfers between the disk and memory
 - We can reduce the number of disk accesses by keeping as many blocks as possible in main memory



Fields, Records, and Files

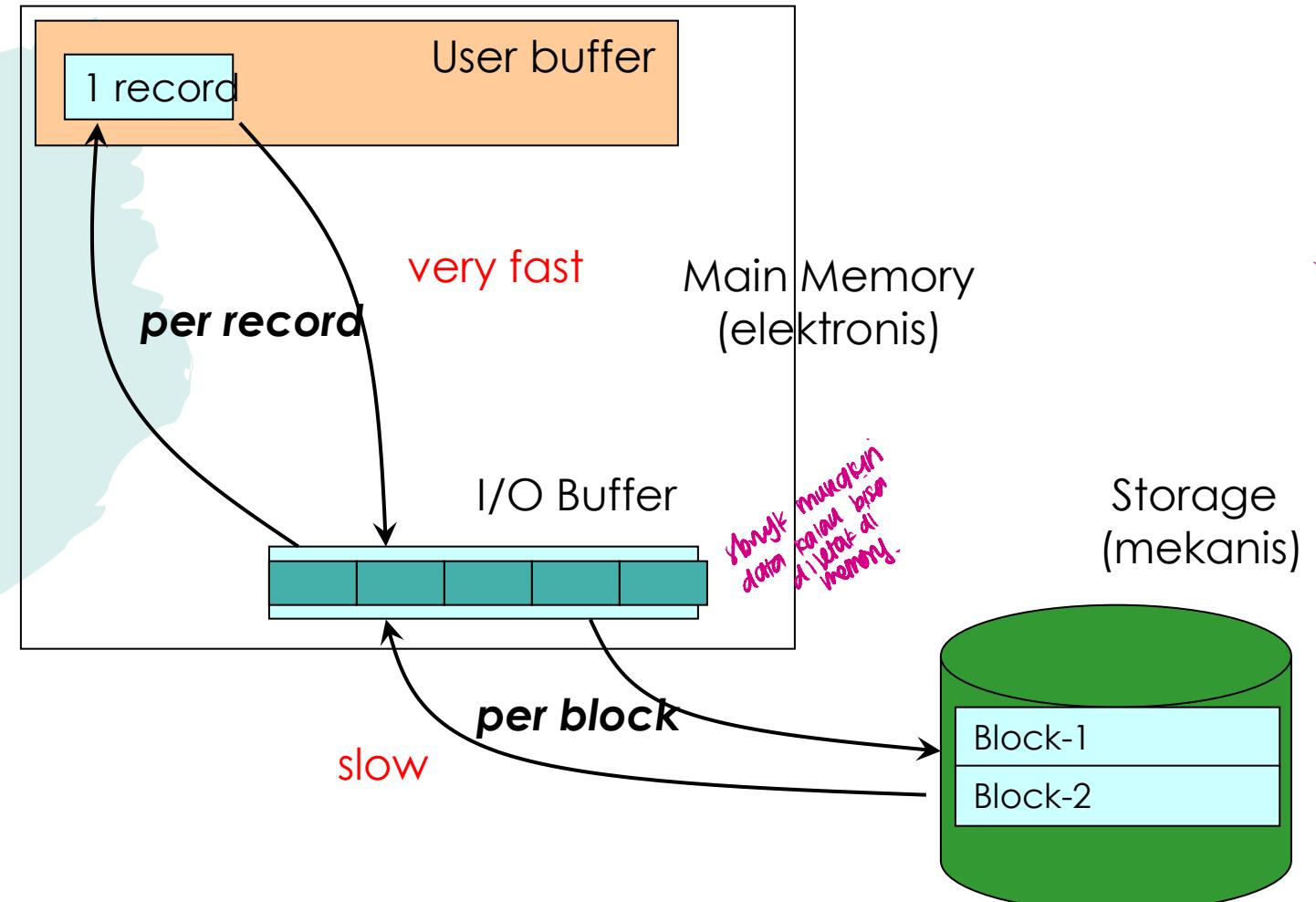


KNOWLEDGE & SOFTWARE ENGINEERING



Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Record Access Mechanism



Optimization of Disk Block Access

- Buffering of blocks
- Disk arm scheduling algorithms
- **File organization**
- Nonvolatile write buffers
- Log disk



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Optimization of Disk Block Access

- **Buffering of blocks** in memory to satisfy future requests
 - **Buffer** – portion of main memory available to store copies of disk blocks.
 - **Buffer manager** – subsystem responsible for allocating buffer space in main memory.
- Programs call on the buffer manager when they need a block from disk.
 - If the block is already in the buffer, buffer manager returns the address of the block in main memory
 - If the block is not in the buffer, the buffer manager:
 - Allocates space in the buffer for the block
 - Replacing (throwing out) some other block, if required, to make space for the new block.
 - Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk.
 - Reads the block from the disk to the buffer and returns the address of the block in main memory to requester

Optimization of Disk Block Access

- **File organization** – optimize block access time by organizing the blocks to correspond to how data will be accessed
 - E.g. Store related information on the same or nearby cylinders.
 - Files may get fragmented over time
 - E.g. if data is inserted to/deleted from the file
 - Or free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
 - Sequential access to a fragmented file results in increased disk arm movement
 - Some systems have utilities to defragment the file system, in order to speed up file access
 - File organization... next.



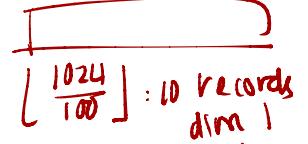
File Organization

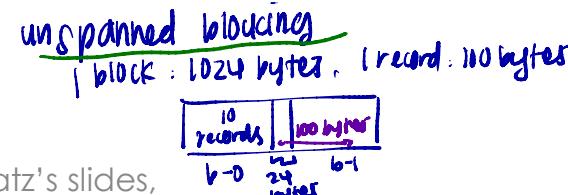


Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

File Organization

- The database is stored as a collection of files. Each file is a sequence of records. A record is a sequence of fields.
- We assume that records are smaller than a disk block
- The fitting of records into blocks is referred to as **blocking**.
 - The blocking can be spanned or unspanned. Unspanned blocking requires all records fit within blocks.
 - **Blocking factor**, denoted by **Bfr**, is the number of records expected within a block
 - For unspanned fixed blocking: $Bfr = \lfloor B/R \rfloor$ where B is the block size and R is the record size

1 record = 100 bytes
1024 bytes

 $\left[\frac{1024}{100} \right] : 10 \text{ records}$
dim 1 block

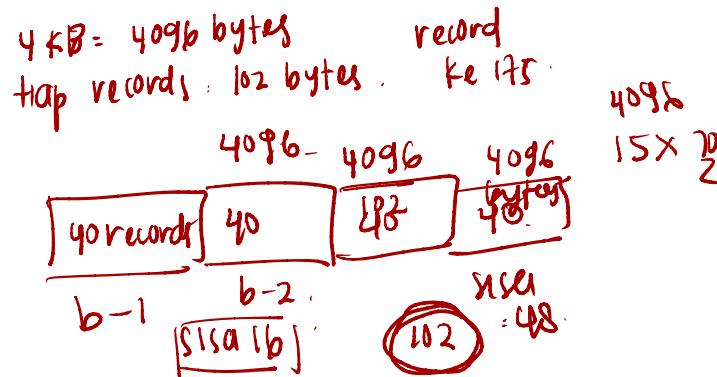


most of the time pacai unspanned..

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Fixed-Length Records

- Simple approach:
 - Store record i starting from byte $n * i$, where n is the size of each record and i starts from 0.
 - Record access is simple, but records may cross blocks
 - Modification: do not allow records to cross block boundaries
- Deletion of record i : alternatives:
 - move records $i + 1, \dots, n$ to $i, \dots, n - 1$
 - move record n to i
 - do not move records, but link all free records on a free list



record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed-Length Records

- Deletion of record i: alternative-1
 - move records $i + 1, \dots, n$ to $i, \dots, n-1$
- Deletion of record i: alternative-2
 - move record n to i

record 0	Record 3 deleted	Kim	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

record 0	Record 3 deleted and replaced by record 11	Kim	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000



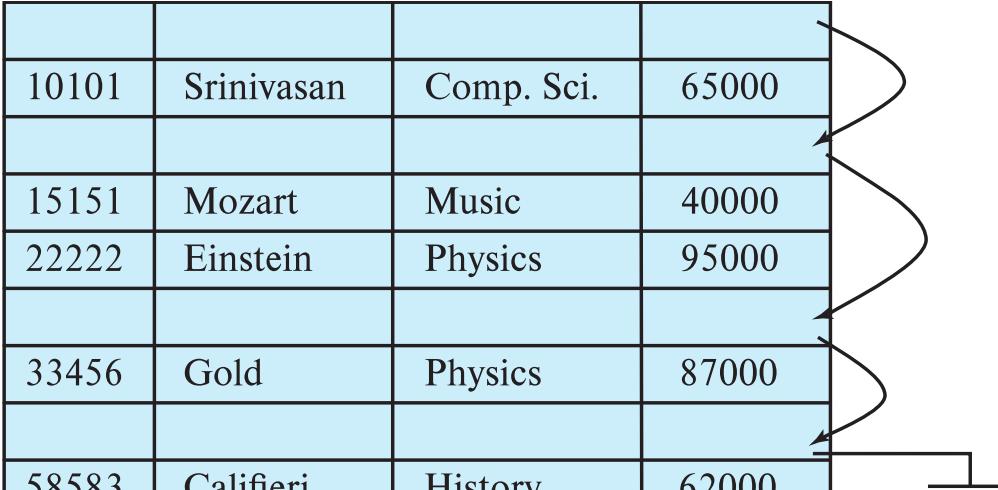
KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Fixed-Length Records

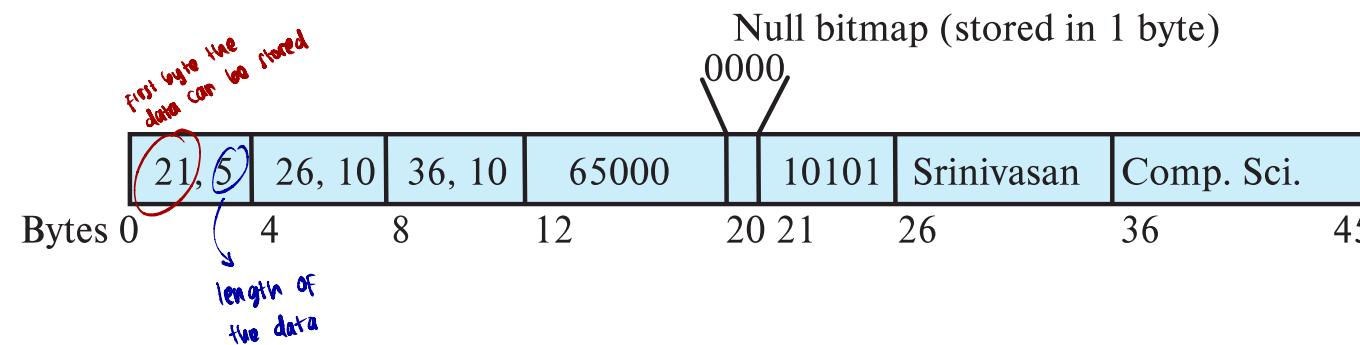
- Deletion of record i:
alternative-3:
 - do not move records, but link all free records on a free list
 - At the beginning of the file, we allocate a certain number of bytes as a file **header**
 - Contains information e.g. the address of the first record whose contents are deleted
 - The deleted records form a linked list often referred to as a **free list**

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000



Variable-Length Records

- Variable-length records arise in database systems in several ways:
 - Storage of multiple record types in a file.
 - Record types that allow variable lengths for one or more fields such as strings (varchar)
 - Record types that allow repeating fields (used in some older data models).
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap

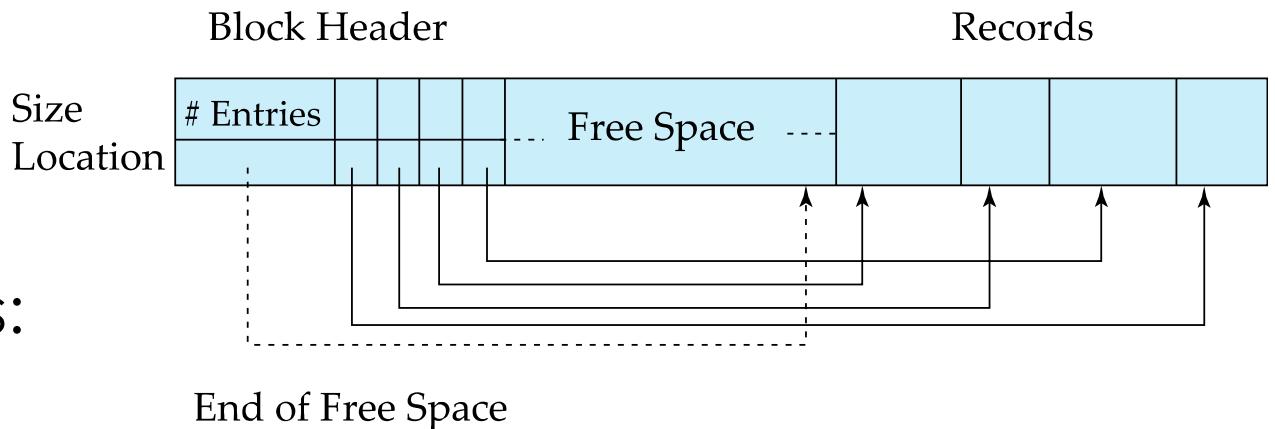


KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Variable-Length Records: Slotted Page Structure

- **Slotted page header** contains:
 - number of record entries
 - end of free space in the block
 - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.



File Performance Parameters

- R : The amount of storage required for a record
- T_F : The time needed to fetch an arbitrary record from the file
- T_N : The time needed to get the next record within the file
- T_I : The time required to update the file by inserting a record
- T_U : The time required to update the file by changing a record
- T_X : The time needed for exhaustive reading of the entire file
- T_Y : The time needed for reorganization of the file

Record Size

- Beside the data, a record may also stores descriptive and access information.
- Descriptive information is advantageous when storing heterogeneous data.

Fetch A Record

To be able to use data from a file, a record containing the data has to be read into the memory.

- Fetch is an associative retrieval of a data element based on a key value.
- Fetching a record consists of 2 steps:
 - Locating the position of the record
 - The actual reading
- In general, the records of a file cannot be directly located on the basis of a subscript value or record number
 - Therefore, a simple address computation cannot be performed

Get the Next Record

Information is mainly generated by relating one fact with another; this implies getting the next record according to some criterion.

- Get-Next is a retrieval using a *structural dependency*.
- A successor record can be obtained most rapidly when related data is kept together; that is, when the locality of these data is strong.
 - When the records are not physically ordered according to the criterion, Get-Next is time consuming.

Insert A Record

Most applications require regular insertion of new data records into their files to remain up to date.

- Inserting a record comprises of:
 - Locating the block where the record will be inserted
 - Reading the block
 - Rewriting the changed block
- When a record has to be put in a specific place within the file, other records may have to be shifted or modified.
- Insertions to the end of the file, Append, are easier to handle.
 - If the address of the last block of the file is known, the performance of this operation can be improved.

Update A Record

All changes of data do not require insertion of a new record.

- When data within a stored record must be changed, the new, updated record is created.
 - The old data and the changes are merged to create a new record,
 - The new record is inserted into the position of the old record within the block, and
 - The block is rewritten into the file.
- If the record has grown in size, it may not be possible to use the old position → **delete** the old record and **insert** a new one.
- Deleting a record is not performed immediately, but instead the record to be deleted is marked with an indicator that the record is now invalid → the delete is now converted to an update.

Reorganize the File

→ misal awalnya organized,
tp setelah several deletion /
insertion bisa
jadi jadi berantakan +
perlu di re-organize.

Reorganization removes deleted and invalidated records, reclaims the space for new data, and restores clustering.

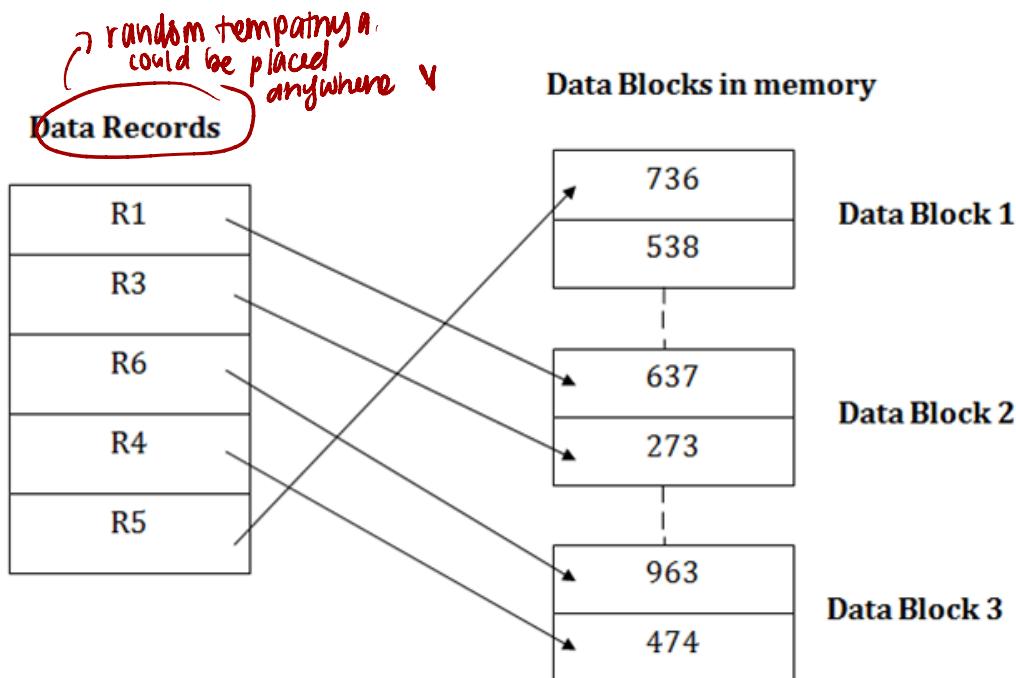
- It is especially important for file organizations which create tombstones for deletion and updates.
- Its frequency depends on the type of file organization used but varies greatly with the application.
- It has many similarities with “garbage collection”.

Organization of Records in Files

- **Heap** – record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- In a **multitable clustering** file organization records of several different relations can be stored in the same file
 - Motivation: store related records on the same block to minimize I/O
- B⁺-tree file organization
 - Ordered storage even with inserts/deletes
 - More on this in next meeting
- Hashing – a hash function computed on search key; the result specifies in which block of the file the record should be placed
 - More on this in next meeting

Heap File Organization

- Heap files are lists of **unordered records** of variable size.
- Records can be placed **anywhere** in the file where there is free space
- Records usually do not move once allocated
- Important to be able to **efficiently find free space** within file



Heap File Organization

- Advantages
 - efficient for bulk loading data
 - efficient for relatively small relations as indexing overheads are avoided
 - efficient when retrievals involve large proportion of stored records
- Disadvantages
 - not efficient for selective retrieval using key values, especially if large
 - sorting may be time-consuming
 - not suitable for volatile tables
 - tabel yg cuma
dibutuhkan sementara,
abis itu dihapus
 - susah for searching, bcs dia acak ↗



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Sequential File Organization

Efficient for system that mostly does sequential data searching

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key
 - Need not be a primary key or even a superkey

Kemudian didalam disk, bukan di dalam memory.

Search-key				
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Sequential File Organization (Cont.)

Literally linked list 😊

- **Deletion** – use pointer chains
- **Insertion** – locate the position where the record is to be inserted
 - if there is free space insert there
 - if no free space, insert the record in an **overflow block**
 - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Jadi kalau insert data yg mau diinsert itu bisa dari block nya beda, tinggal pointer aja

32222	Verdi	Music	48000	
-------	-------	-------	-------	--

Multitable Clustering File Organization

↳ kalau 2 data sering di join,
pakai multitable clustering biar dalam
1 block.

- Store several relations in one file using a **multitable clustering** file organization
- good for queries involving:
 - department \bowtie instructor, and
 - one single department and its instructors
- bad for queries involving only department → karena harus ambil pdhl 1 block butuh department doang.
- results in variable size records
- Can add pointer chains to link records of a particular relation

department

dept_name	building	budget
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

Data Dictionary Storage

- The **Data dictionary** (also called **system catalog**) stores metadata; that is, data about data, such as
 - Information about relations
 - names of relations
 - names, types and lengths of attributes of each relation
 - names and definitions of views
 - integrity constraints
 - User and accounting information, including passwords
 - Statistical and descriptive data
 - number of tuples in each relation
 - Physical file organization information
 - How relation is stored (sequential/hash/...)
 - Physical location of relation
 - Information about indices (next meeting)

Relational Representation of System Metadata

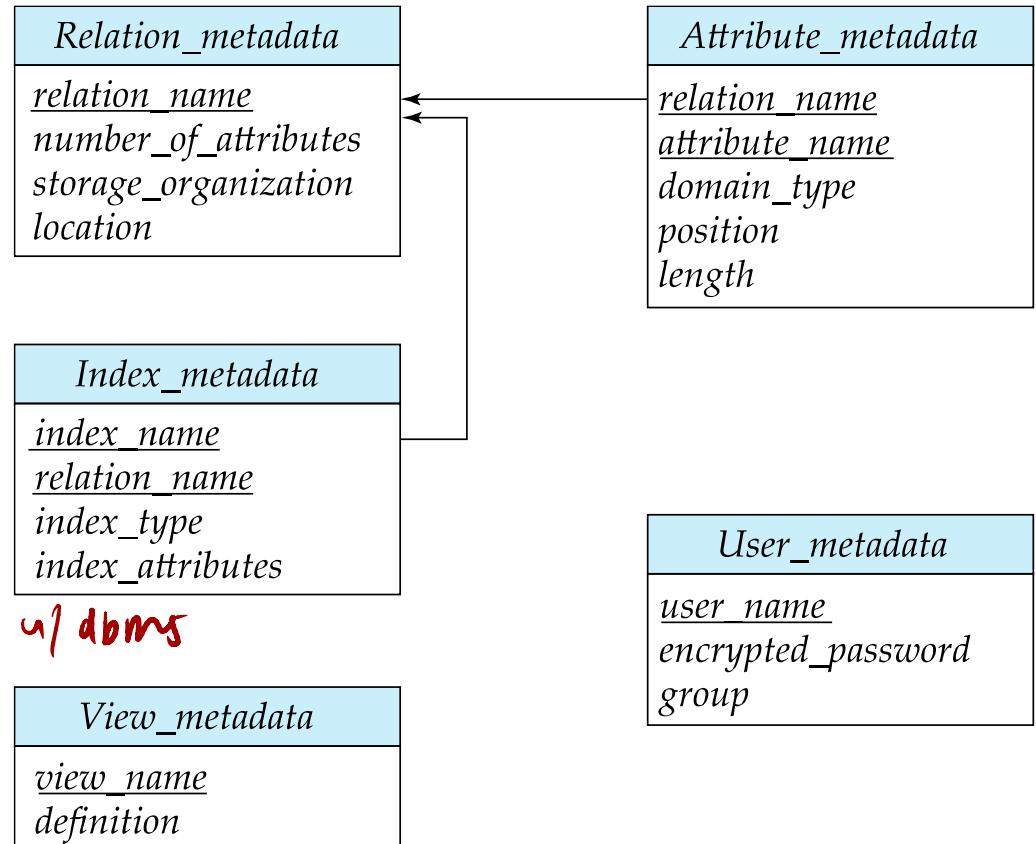
→ merupakan relasi ke dalam relasi

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory

information schema

↳ punya > 1 view yg data bisa diambil u/ dbms

ex: select * from information-schema.tables





Additional Materials

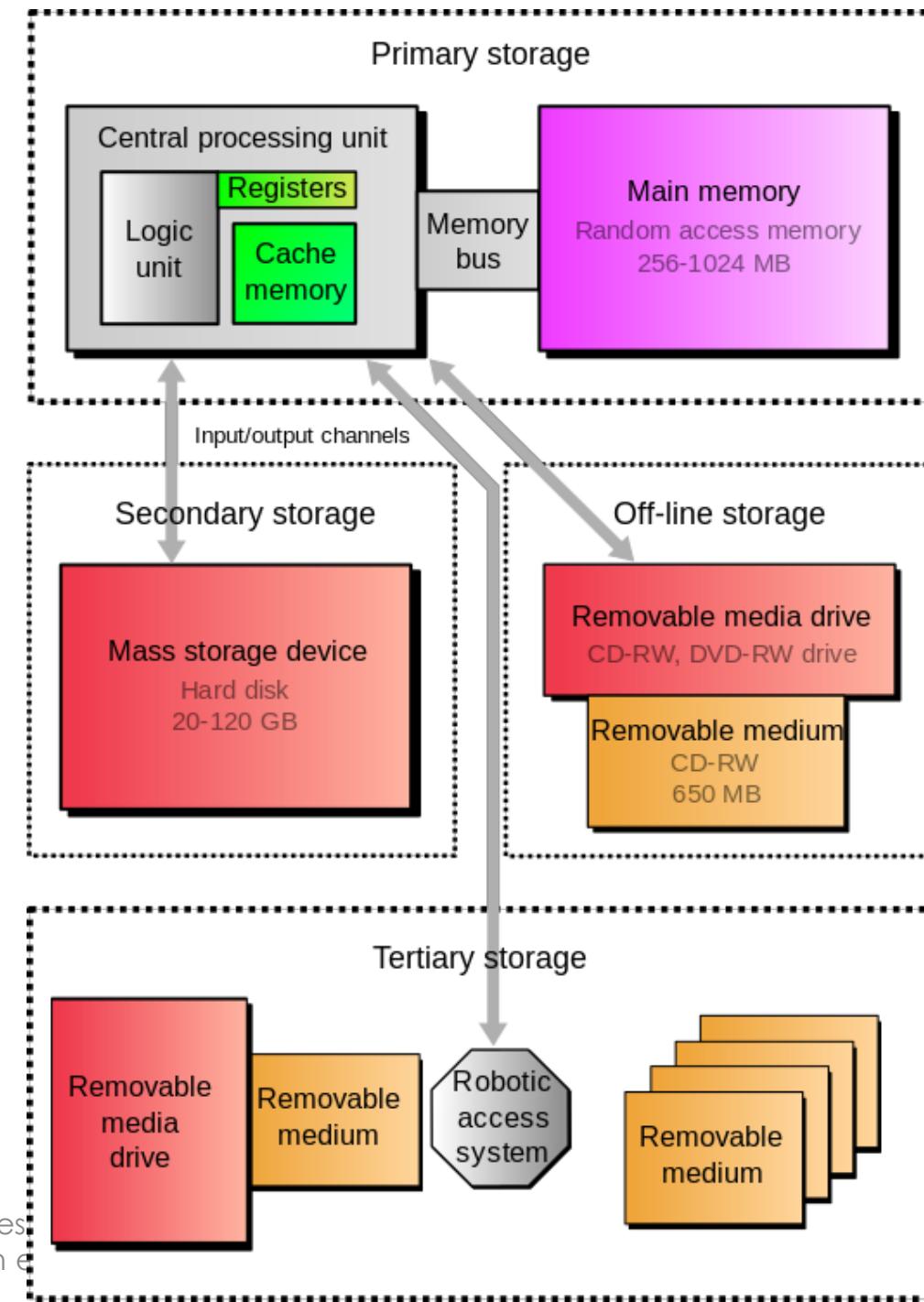


KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Storage Hierarchy

- **primary storage:** Fastest media but volatile (cache, main memory).
- **secondary storage:** next level in hierarchy, non-volatile, moderately fast access time
 - Also called **on-line storage**
 - E.g. flash memory, magnetic disks
- **tertiary storage:** lowest level in hierarchy, non-volatile, slow access time
 - also called **off-line storage** and used for archival storage
 - e.g., magnetic tape, optical storage
 - Magnetic tape
 - Sequential access, 1 to 12 TB capacity
 - A few drives with many tapes
 - Juke boxes with petabytes (1000's of TB) of storage

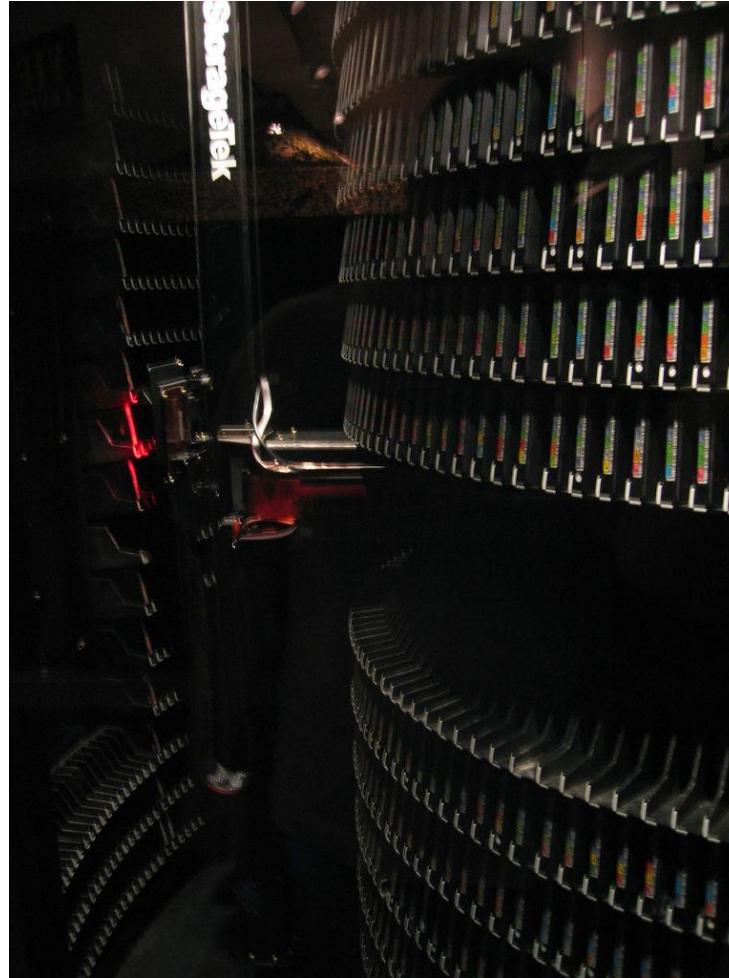




A hard disk drive with protective cover removed
By Evan-Amos - Own work, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=27941467>



KNOWLEDGE & SOFTWARE ENGINEERING

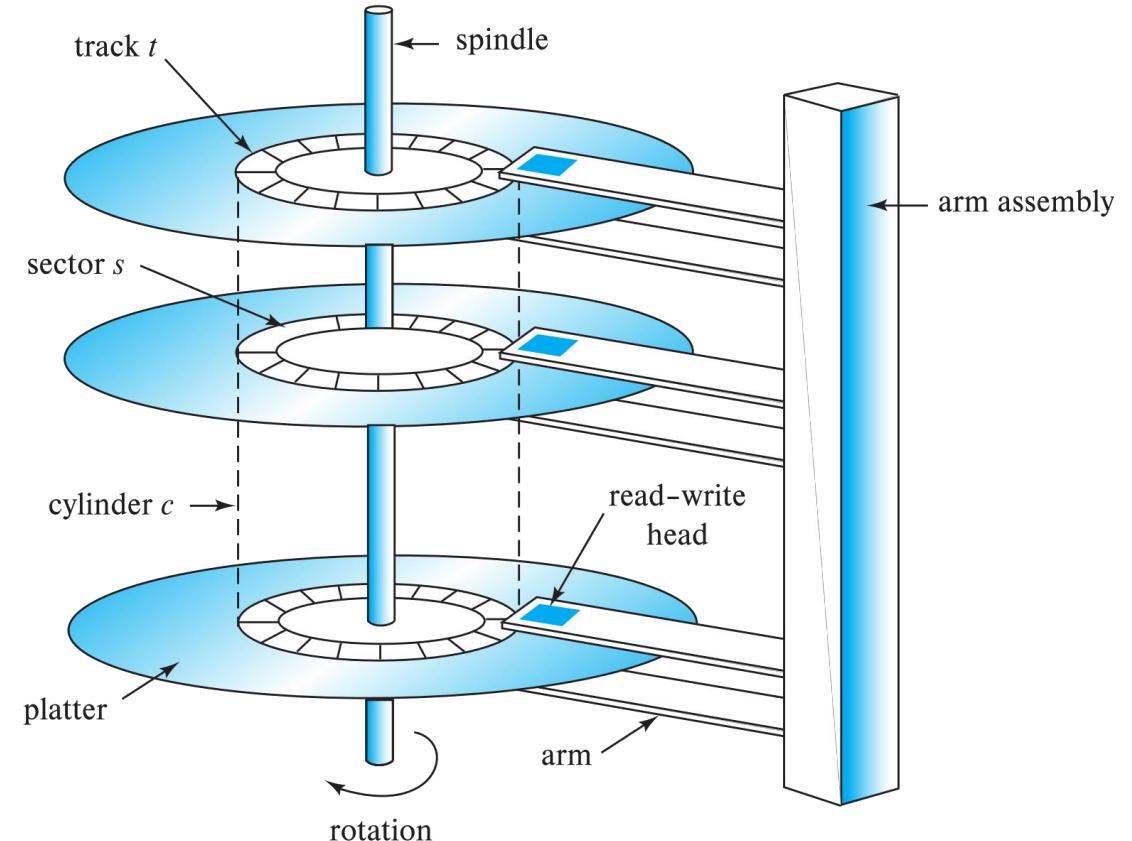


A large tape library, with tape cartridges placed on shelves in the front, and a robotic arm moving in the back. Visible height of the library is about 180 cm.
Austin Mills from Austin, TX, USA [CC BY-SA]
(<https://creativecommons.org/licenses/by-sa/2.0/>)

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Magnetic Hard Disk Mechanism

- Surface of platter divided into circular **tracks**
- Each track is divided into **sectors**
 - A sector is the smallest unit of data that can be read or written
- Read-Write Head
- Head-disk assemblies
- **Cylinder i** consists of i -th track of all the platters

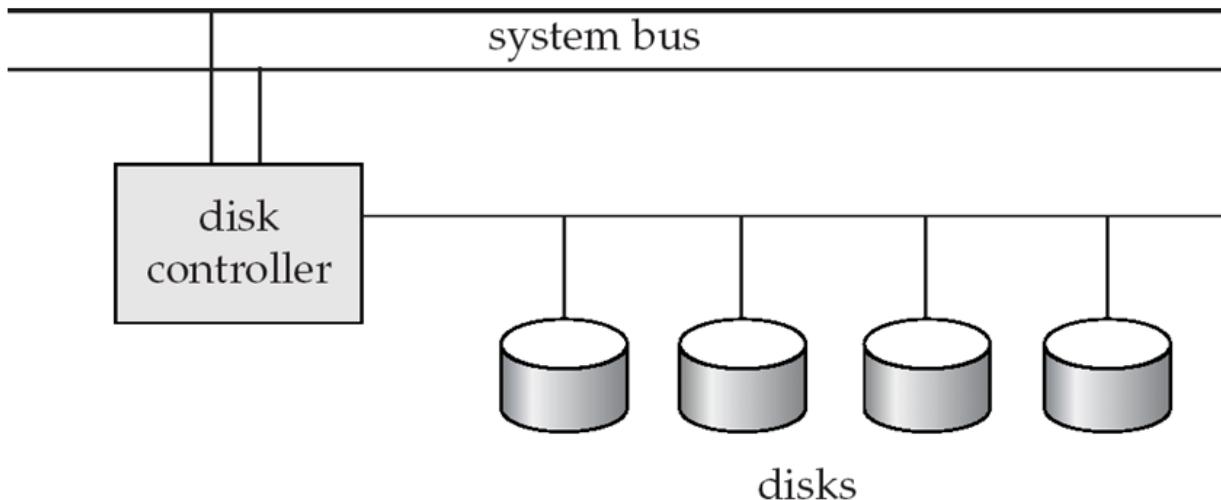


Schematic diagram of magnetic disk drive

Magnetic Disks

- **Disk controller** – interfaces between the computer system and the disk drive hardware.
 - accepts high-level commands to read or write a sector
 - initiates actions such as moving the disk arm to the right track and actually reading or writing the data
 - Computes and attaches **checksums** to each sector to verify that data is read back correctly
 - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
 - Ensures successful writing by reading back sector after writing it
 - Performs **remapping of bad sectors**

Disk Subsystem



- Multiple disks connected to a computer system through a disk controller
 - Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller
- Disk interface standards families
 - **ATA** (AT Attachment) range of standards
 - **SATA** (Serial ATA)
 - **SCSI** (Small Computer System Interconnect) range of standards
 - **SAS** (Serial Attached SCSI)
 - Several variants of each standard (different speeds and capabilities)

Performance Measures of Disks

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins, consists of:
 - **Seek time** – time it takes to reposition the arm over the correct track
 - Average seek time is $1/2$ the worst case seek time.
 - Would be $1/3$ if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
 - 4 to 10 milliseconds on typical disks
 - **Rotational latency** – time it takes for the sector to be accessed to appear under the head
 - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
 - Average latency is $1/2$ of the above latency.
 - Overall latency is 5 to 20 msec depending on disk model
 - **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk
 - 25 to 200 MB per second max rate, lower for inner tracks

Performance Measures of Disks

- **Mean time to failure (MTTF)** – the average time the disk is expected to run continuously without any failure.
 - Typically 3 to 5 years
 - Probability of failure of new disks is quite low, corresponding to a “theoretical MTTF” of 500,000 to 1,200,000 hours for a new disk
 - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours, but does not imply that the disks are expected to function 136 years!
 - MTTF decreases as disk ages

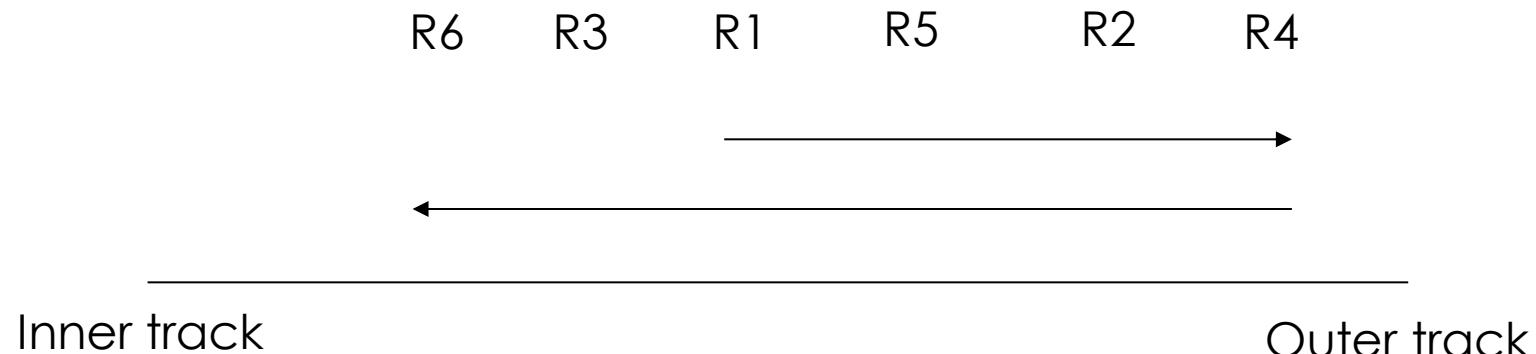


KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
“Database System Concepts”, 7th ed.

Optimization of Disk Block Access

- **Disk-arm-scheduling algorithms** order pending accesses to tracks so that disk arm movement is minimized
 - elevator algorithm:



Optimization of Disk Block Access

- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM buffer immediately
 - Non-volatile RAM (NV-RAM): battery backed up RAM or flash memory
 - Even if power fails, the data is safe and will be written to disk when power returns
 - Controller then writes to disk whenever the disk has no other requests or request has been pending for some time
 - Database operations that require data to be safely stored before continuing can continue without waiting for data to be written to disk
 - Writes can be reordered to minimize disk arm movement
- **Log disk** – a disk devoted to writing a sequential log of block updates
 - Used exactly like nonvolatile RAM
 - Write to log disk is very fast since no seeks are required
 - No need for special hardware (NV-RAM)
- File systems typically reorder writes to disk to improve performance
 - **Journaling file systems** write data in safe order to NV-RAM or log disk
 - Reordering without journaling: risk of corruption of file system data

IF3140 – Sistem Basis Data Indexing

SEMESTER II TAHUN AJARAN 2024/2025



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from [Silberschatz's slides](#),
“Database System Concepts”, 7th ed.



Modified from Silberschatz's slides,
“Database System Concepts”, 7th ed.

Sumber

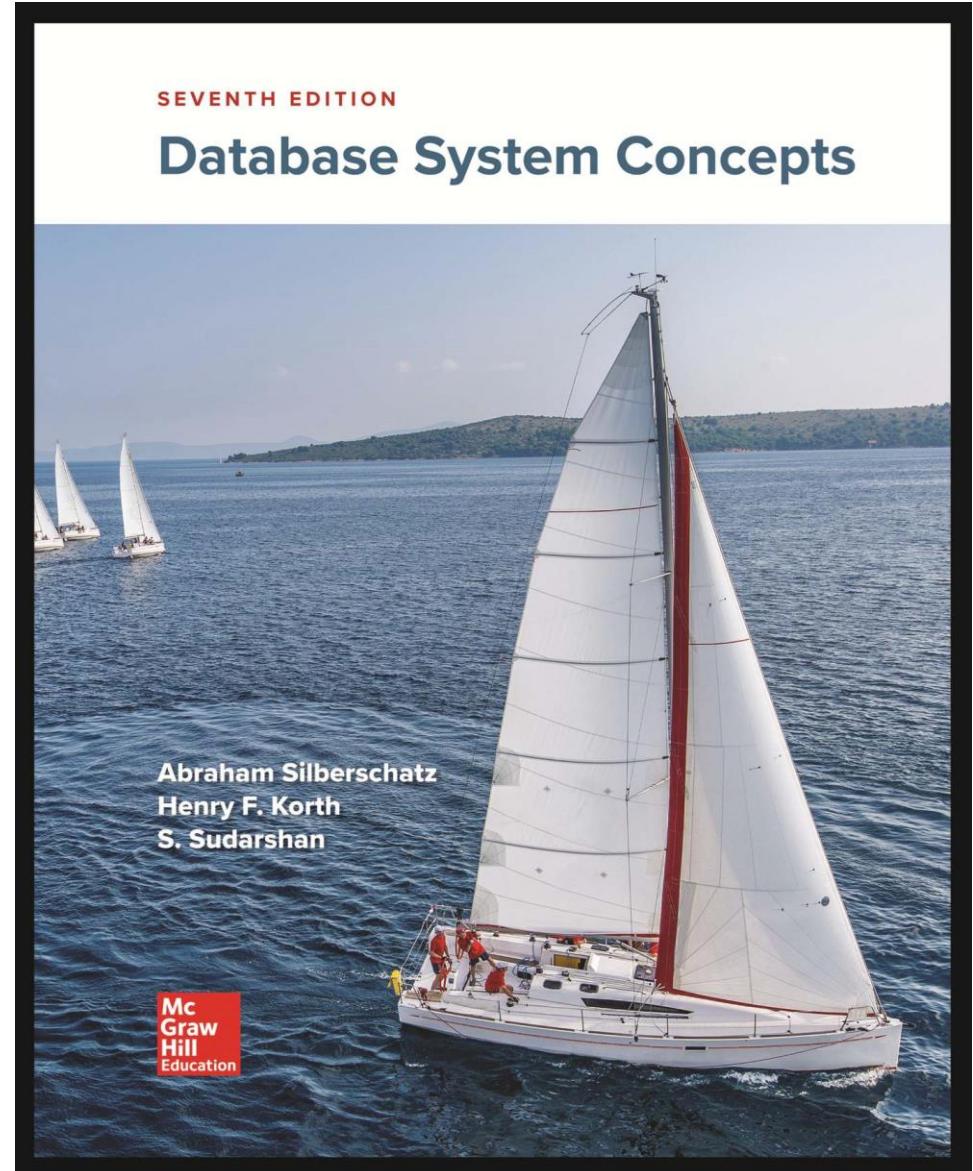
Silberschatz, Korth, Sudarshan:
“Database System
Concepts”, 7th Edition

- **Chapter 14:** Indexing



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
“Database System Concepts”, 7th ed.



index terdiri dari records yg terdiri dari:

Index

records

search key
pointer

- Index → used to speed up access to desired data.
 - E.g., author catalog in library
- **Search Key** - attribute or set of attributes used to look up records in a file.
- An **index file** consists of records (called **index entries**) of the form



pointing untuk ke index berikutnya/record

- Two basic kinds of indices:
 - **Ordered indices**: search keys are stored in sorted order
 - **Hash indices**: search keys are distributed uniformly across “buckets” using a “hash function”.

tidak depends on keturunan indexnya,
melainkan dari hash functionnya.

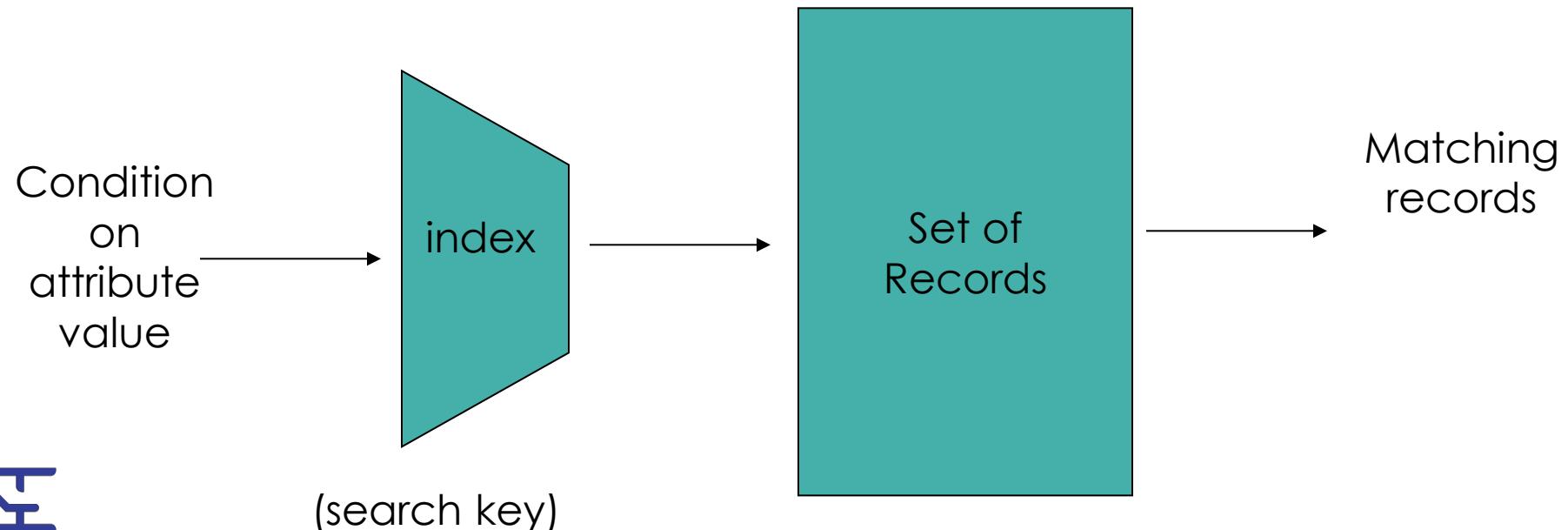


KNOWLEDGE & SOFTWARE ENGINEERING

Index

→ primary key is automatically indexed, sisanya manual by user

- An index is a data structure that **supports efficient access** to data
- It facilitates searching, sorting, join operation, etc., efficiently instead of scanning all table rows
- Index files are typically **much smaller** than the original file



Index Evaluation Metrics

- Access types supported efficiently. E.g.,
 - records with a **specified value** in the attribute where id = 1
 - or records with an attribute value falling in a specified **range of values**. where id > 1 and id < 10
- Access time → time to get the data / when the data has index (query read)
especially
- Insertion time → time taken when we insert new record to the table · ↗ may improve waktunya baca tp memperlambat insert/delete.
- Deletion time → time taken to delete a record.
- Space overhead
 - ↳ index is a new file so need more space.



KNOWLEDGE & SOFTWARE ENGINEERING

Several Index Methods

- Several index methods we will discuss:
 - Ordered indices ([index-sequential file](#))
 - B⁺-tree
 - Hash indexes
 - Bitmap indexes



KNOWLEDGE & SOFTWARE ENGINEERING

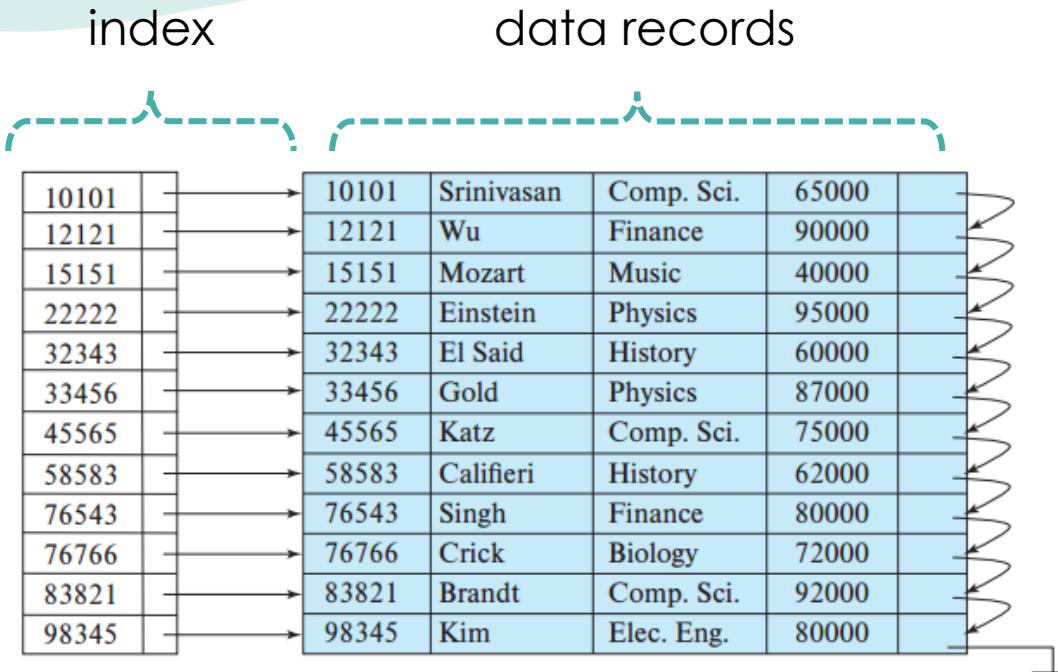
Ordered Indices

ada separate file of index yg contain value based on the search key.

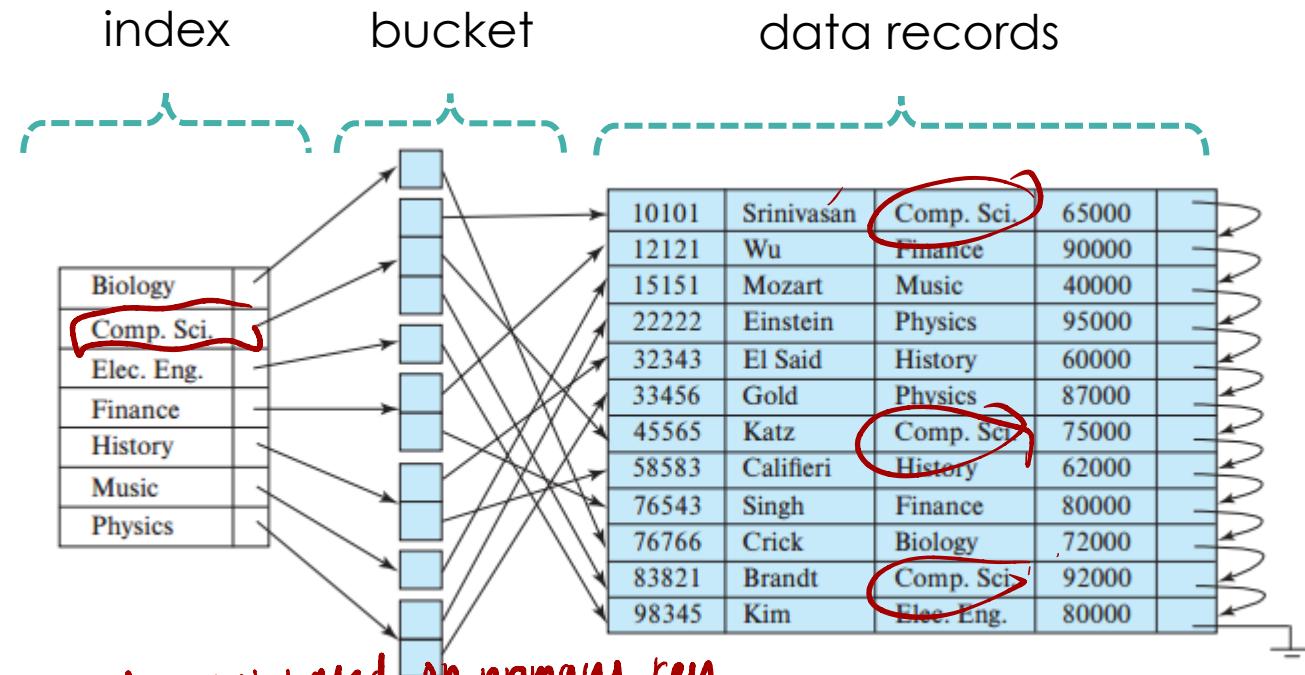
- In an **ordered index**, index entries are stored sorted on the search key value. E.g., author catalog in library.
- **Primary index:** an index whose search key specifies the sequential order of the file. Also called clustering index.
 - A table can **have only one clustered index**
 - The search key of a primary index is **usually** but not necessarily the primary key.
urutan index menentukan urutan physical
- **Secondary index:** an index whose search key specifies an order different from the sequential order of the file. Also called non-clustering index.
search key of index ≠ search key of file.
urutan index tidak menentukan urutan physical
- Index-sequential file: ordered sequential file with a primary index.



Primary (clustering) vs. Secondary (non-clustering) Index



Primary index (clustering index) on **ID**



Secondary index (non-clustering index) on **dept_name**

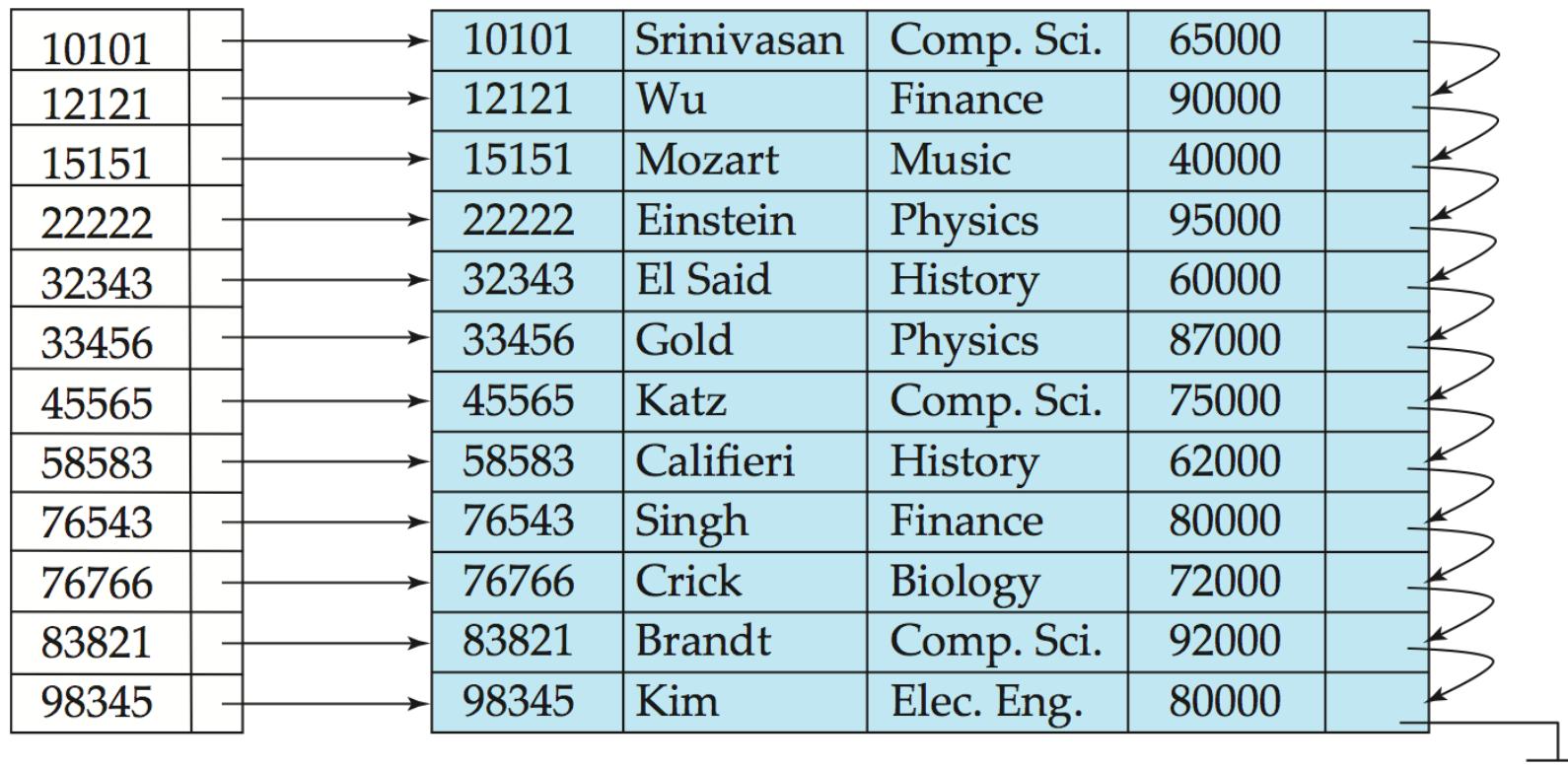
- Thus, a table can **have only one clustered index**
- E.g., SQL Server 2005 supports up to 249 non-clustering indices on a table

Ordered indices - Dense Index

semua value yg ada
di atribut search key
ada di index

- **Dense index** — Every search-key appears in the index records.
- E.g. index with search key on **ID** attribute of **instructor** relation

1 search key menunjuk 1 record



Note:

- Record is sorted based on the search key in a linked-list structure
- Locating index record is done by binary search (complexity = $\log_2 n$)

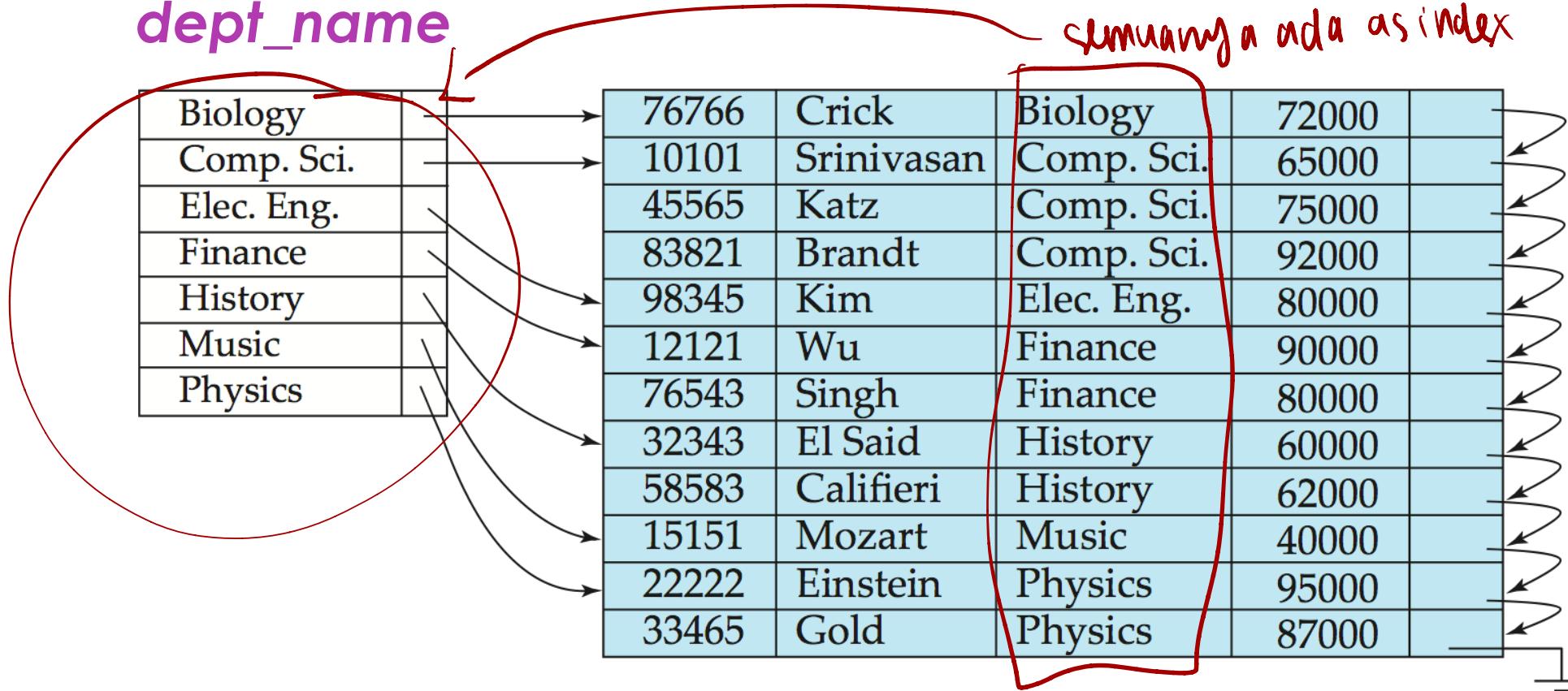


KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

Ordered indices - Dense Index (Cont.)

- Dense index on **dept_name**, with *instructor* file **sorted on dept_name**



See! All the search-keys appear in the index file

Ordered indices – Sparse Index

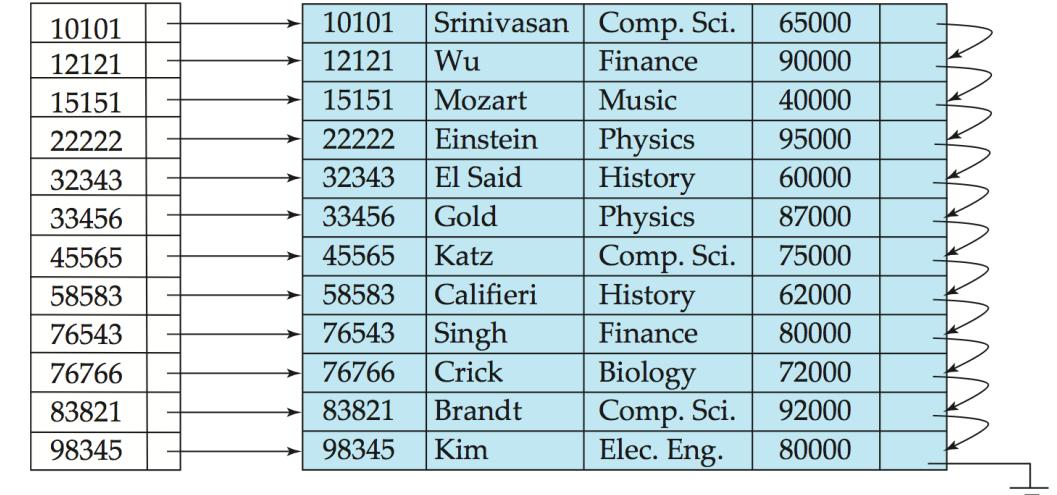
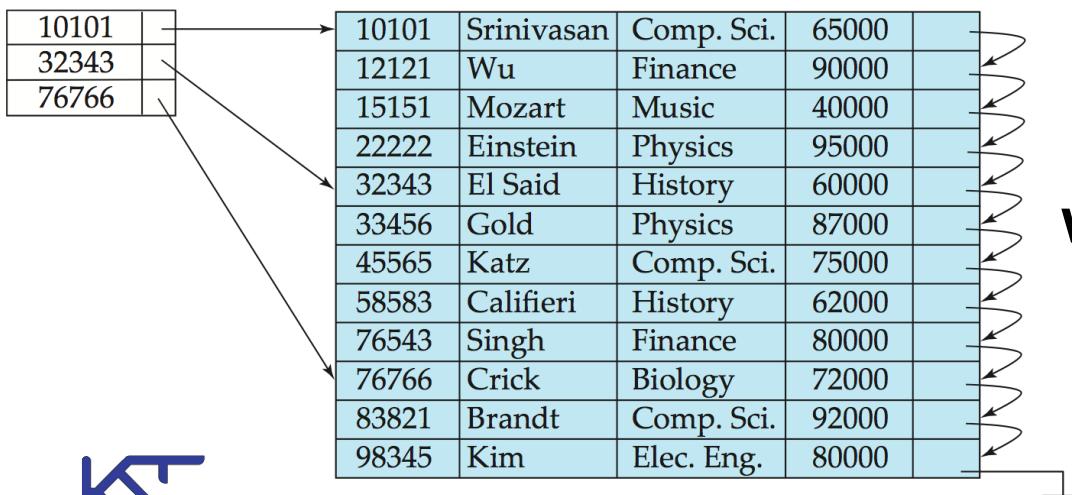
- **Sparse Index:** only some search-key values appear in the index records.
 - Applicable when **records are sequentially ordered on search-key**
- To locate a record with search-key value **K**, we:
 - Find index record with **largest search-key value that is $\leq K$**
 - **Search file sequentially** starting at the record to which the index record points

value palin
besar dari yg
lebih kecil dari
yg kita cari
mau yg ga
ada di search
key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	E. Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

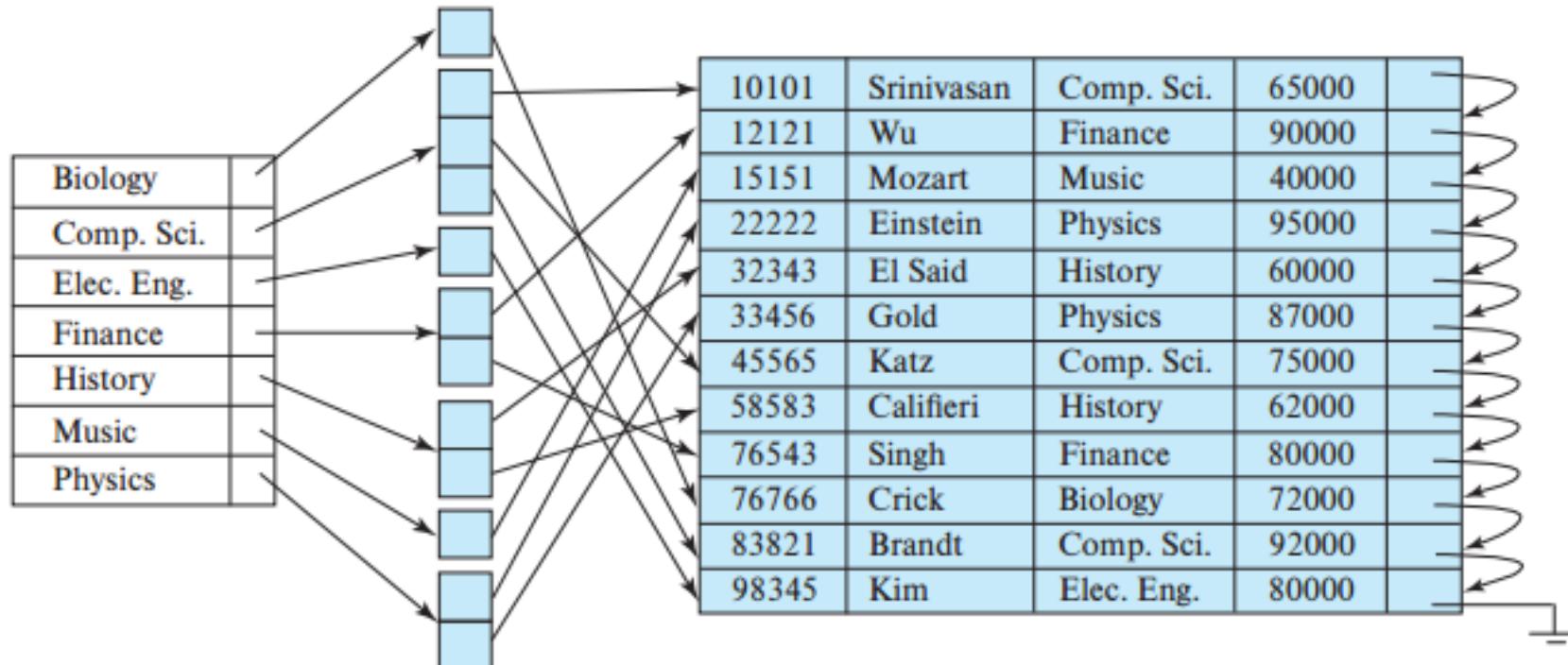
Ordered indices – Sparse Index (Cont.)

- Compared to dense indices:
 - Less space and less maintenance overhead for insertions and deletions.
 - Generally slower than dense index for locating records.
- **Good tradeoff:** sparse index with an index entry for every block in file, corresponding to least search-key value in the block.



Ordered indices – Secondary Index (revisit)

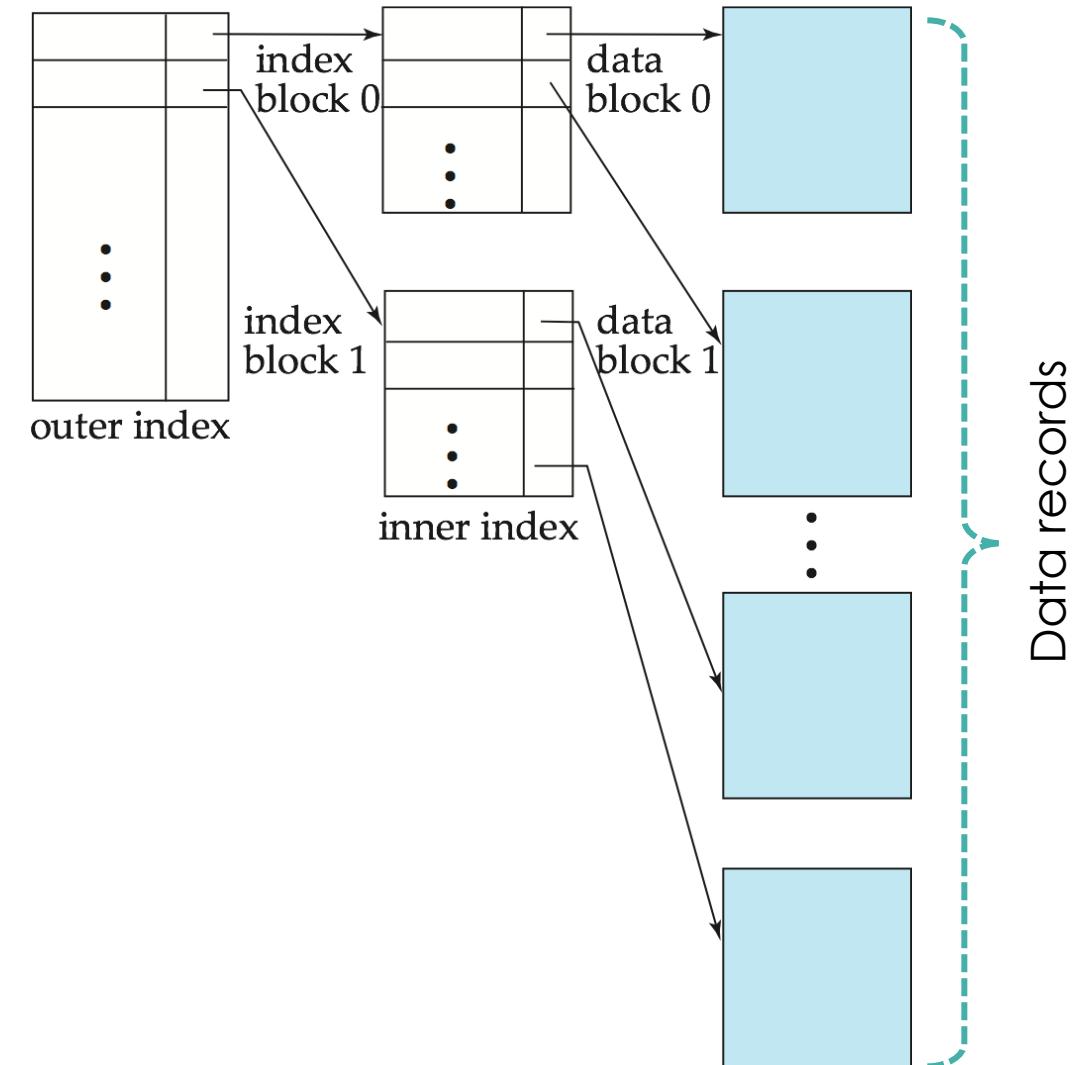
- Index record **points to a bucket** that contains pointers to all the actual records with that particular search-key value.
- Secondary indices **have to be dense**



Secondary index on `dept_name` attribute of `instructor`

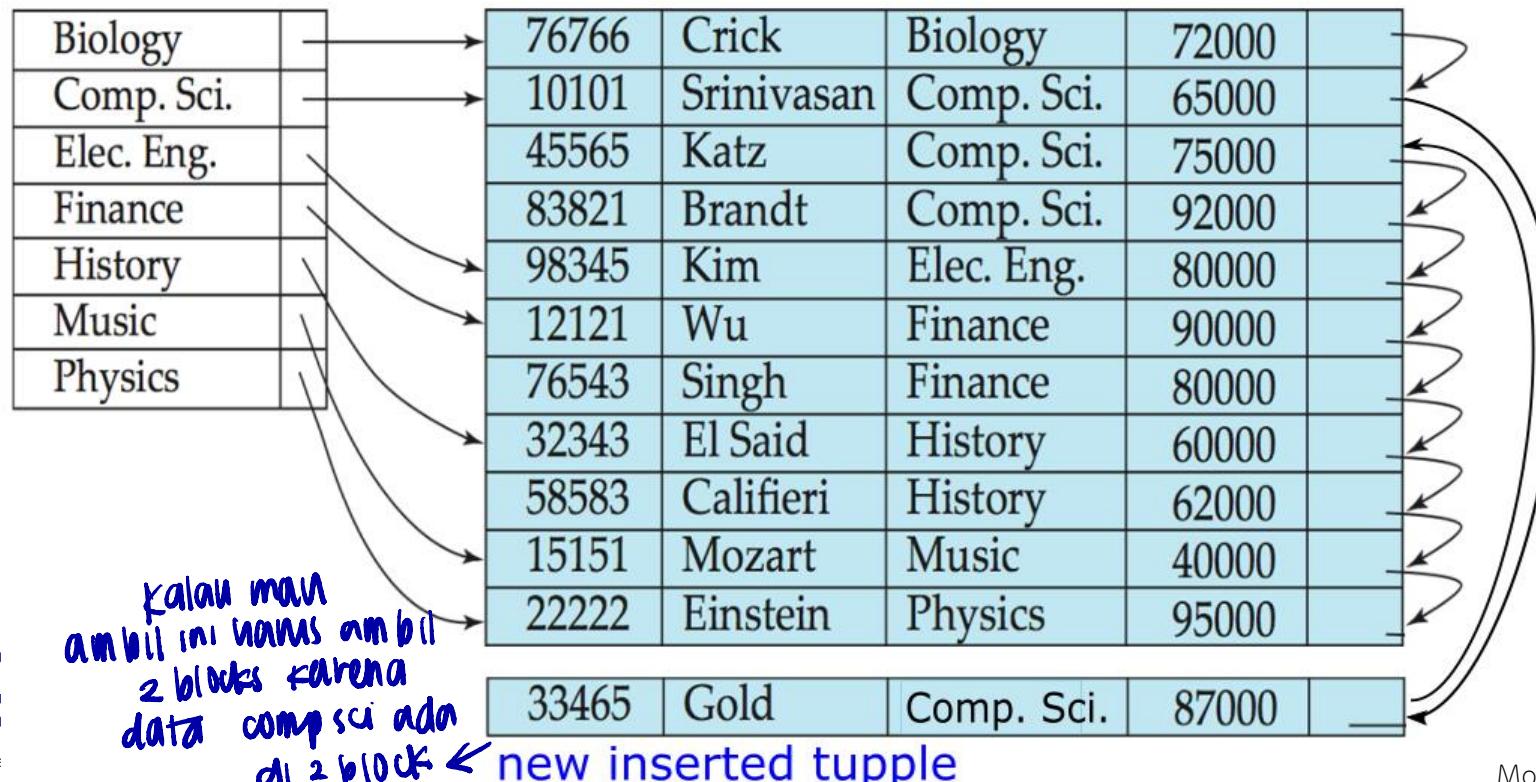
Ordered indices – Multilevel Index

- If primary index does not fit in memory, access becomes expensive. → *index also file stored in blocks*
- **Solution:** treat primary index kept on disk as a sequential file and construct a sparse index on it.
 - **outer index** – a sparse index of primary index
 - **inner index** – the primary index file



Ordered indices on Insert, Delete, Update

- In **index-sequential file**, as file grows with insert, delete and update, many **overflow blocks get created**.
- **Periodic reorganization** of entire file is required, otherwise, **more random disk I/O is needed**.



kalan misal dia dih aurut jd gaperlu 2 blocks karena compsi nya dah keurut.

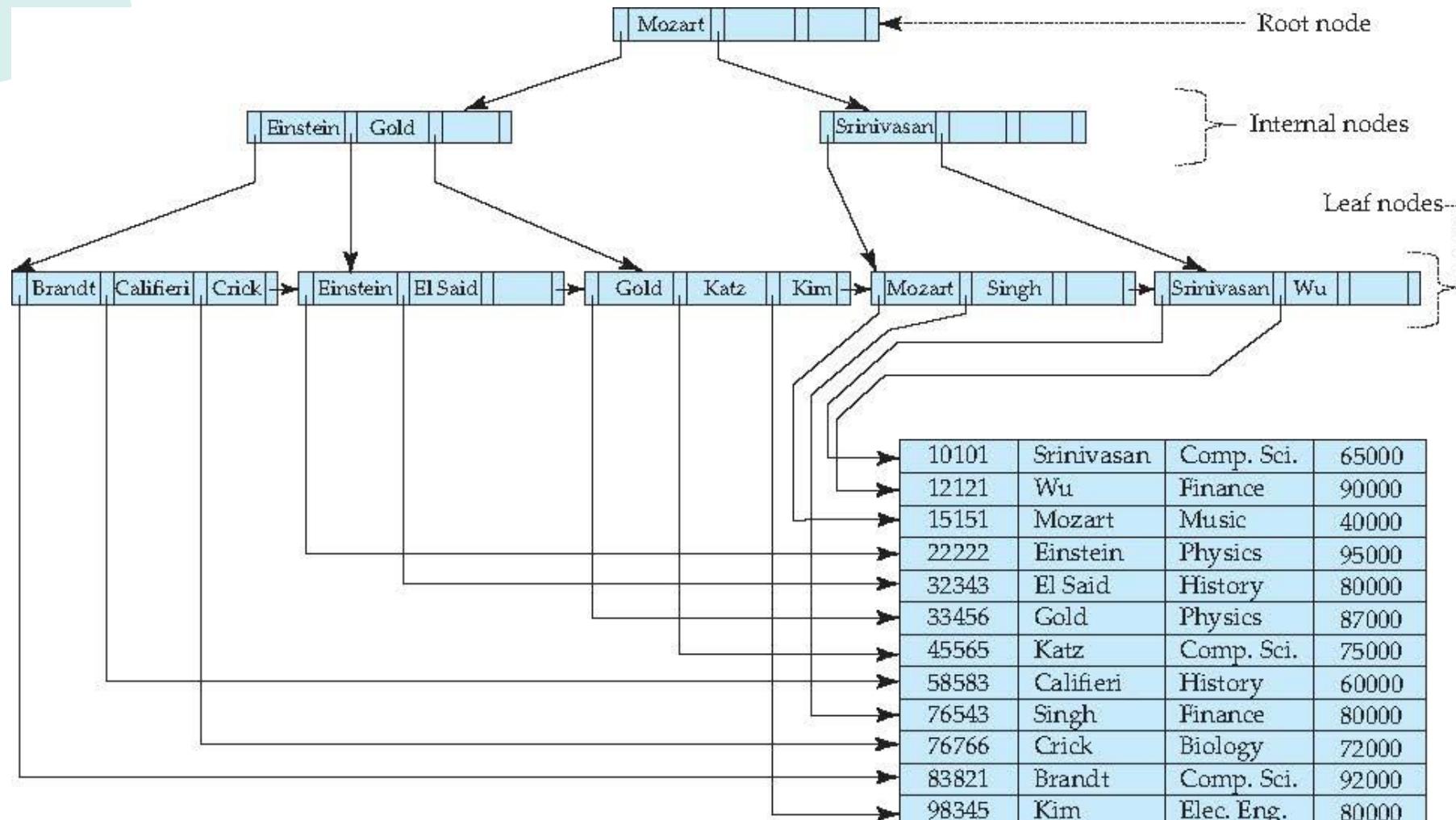
E.g., querying records whose **dept_name** is "Comp. Sci." now needs more disk I/O

B⁺-Tree Index

B⁺-tree indices are **an alternative to indexed-sequential files**.

- Disadvantage of indexed-sequential files
 - **performance degrades** as file grows, since **many overflow blocks get created**.
 - **Periodic reorganization** of entire file is required.
 - Searching using binary search has **(log₂ n) cost**, later, we will know a cheaper cost using B⁺-tree
- Advantage of B⁺-tree index files:
 - automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
 - **Reorganization of entire file is not required** to maintain performance.
- (Minor) disadvantage of B⁺-trees:
 - extra insertion and deletion overhead, space overhead.
- Advantages of B⁺-trees **outweigh** disadvantages
 - **B⁺-trees are used extensively**

Example of B^+ -Tree



B+-Tree Index (Cont.)

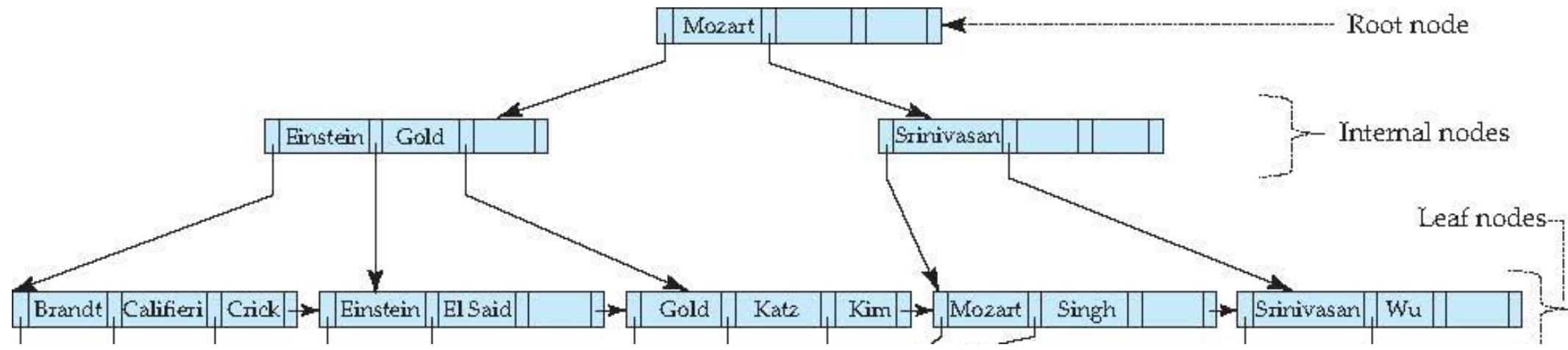
- A B+-tree with max degree n is a rooted tree satisfying the following properties:
 - All paths from root to leaf are of the same length
 - Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children
 - A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
 - Special cases:
 - If the root is not a leaf, it has at least 2 children.
 - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n-1)$ values.



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

B+-Tree Index (Cont.)

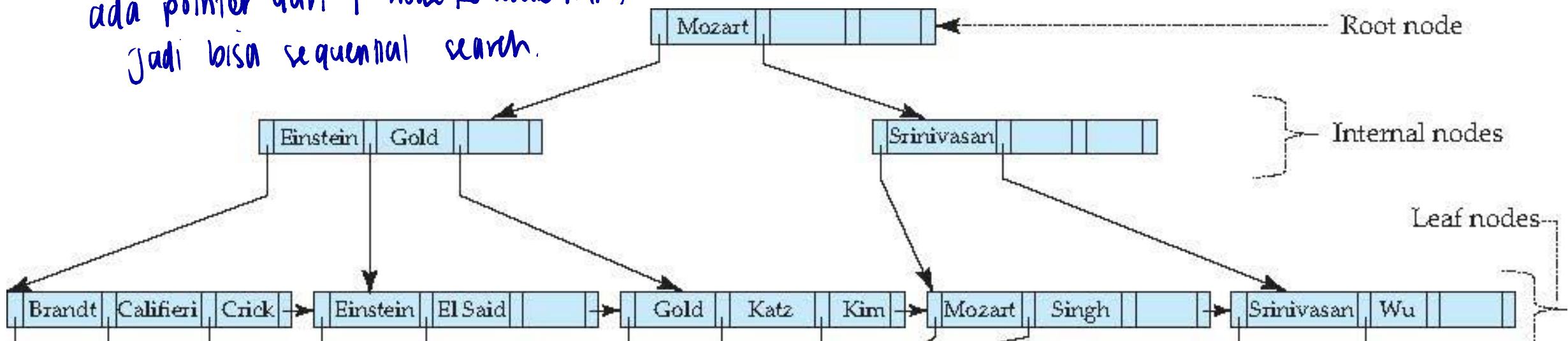


A B⁺-tree example with $n = 4$

B⁺-Tree Index (Cont.)

- Search **key is in sorted order**, having properties of n-ary search tree
- There is a **pointer** in the end of leaf pointing to the **next leaf**
- The non-leaf levels of the B⁺-tree form a hierarchy of sparse indices.

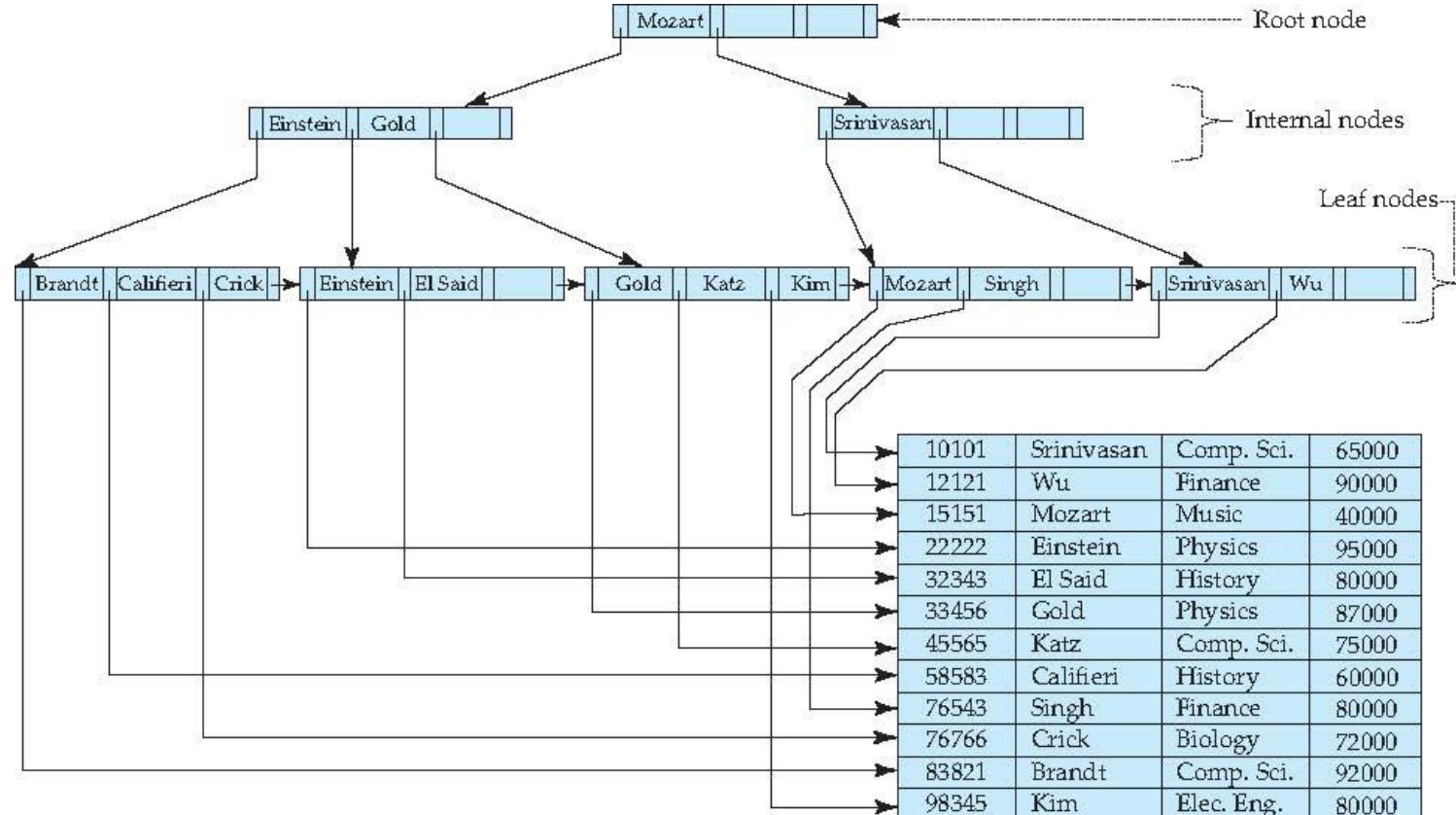
internal node
ada pointer dari 1 node ke node lain,
jadi bisa sequential search.



A B⁺-tree example with $n = 4$

Queries on B^+ -Trees

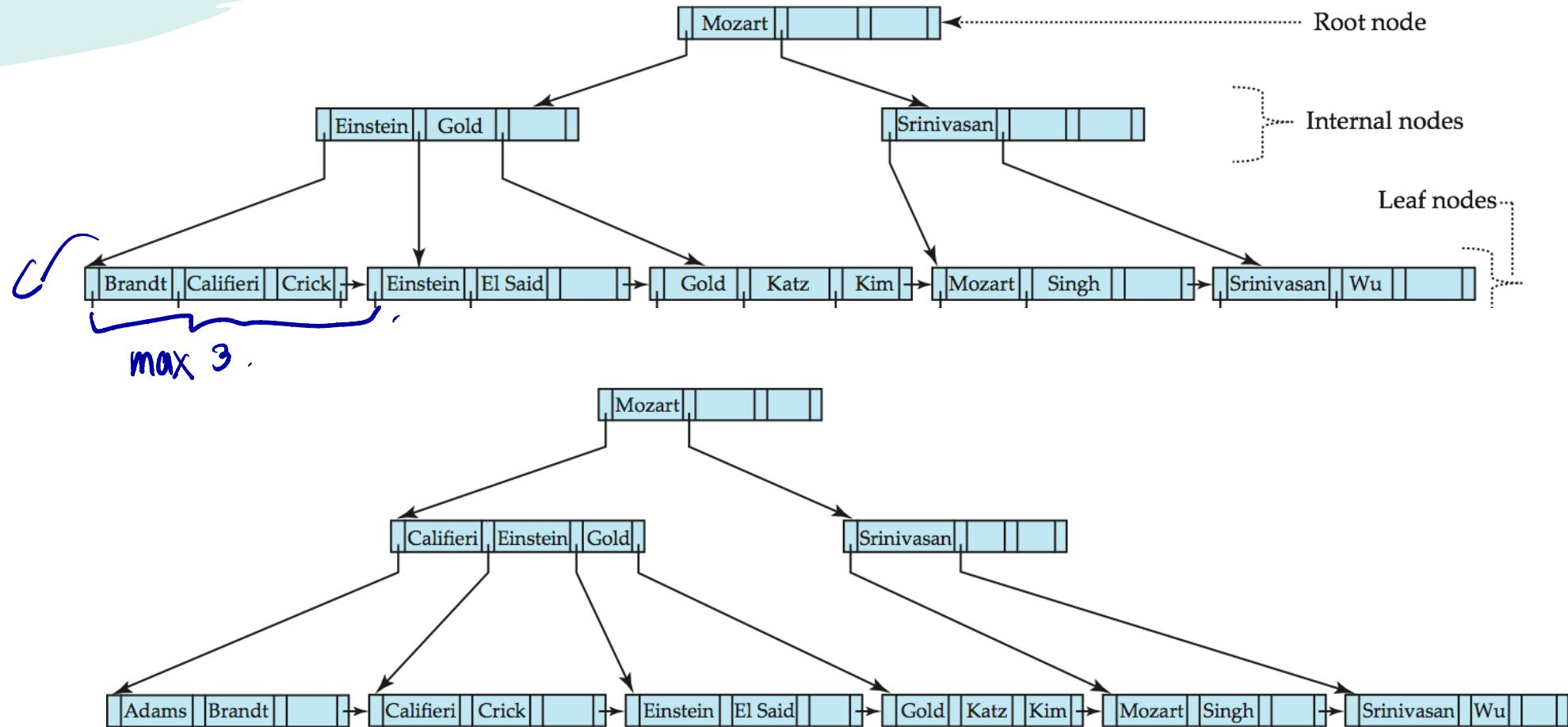
- With K search-key values, the tree height is no more than $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$.
- A node is generally the same size as a disk block, typically 4 KB
 - and n is typically around 100 (40 bytes per index entry).
- With 1 million search key values and $n = 100$
 - at most $\log_{50}(1,000,000) = 4$ nodes are accessed in a lookup.



- It handles efficiently **range query!**

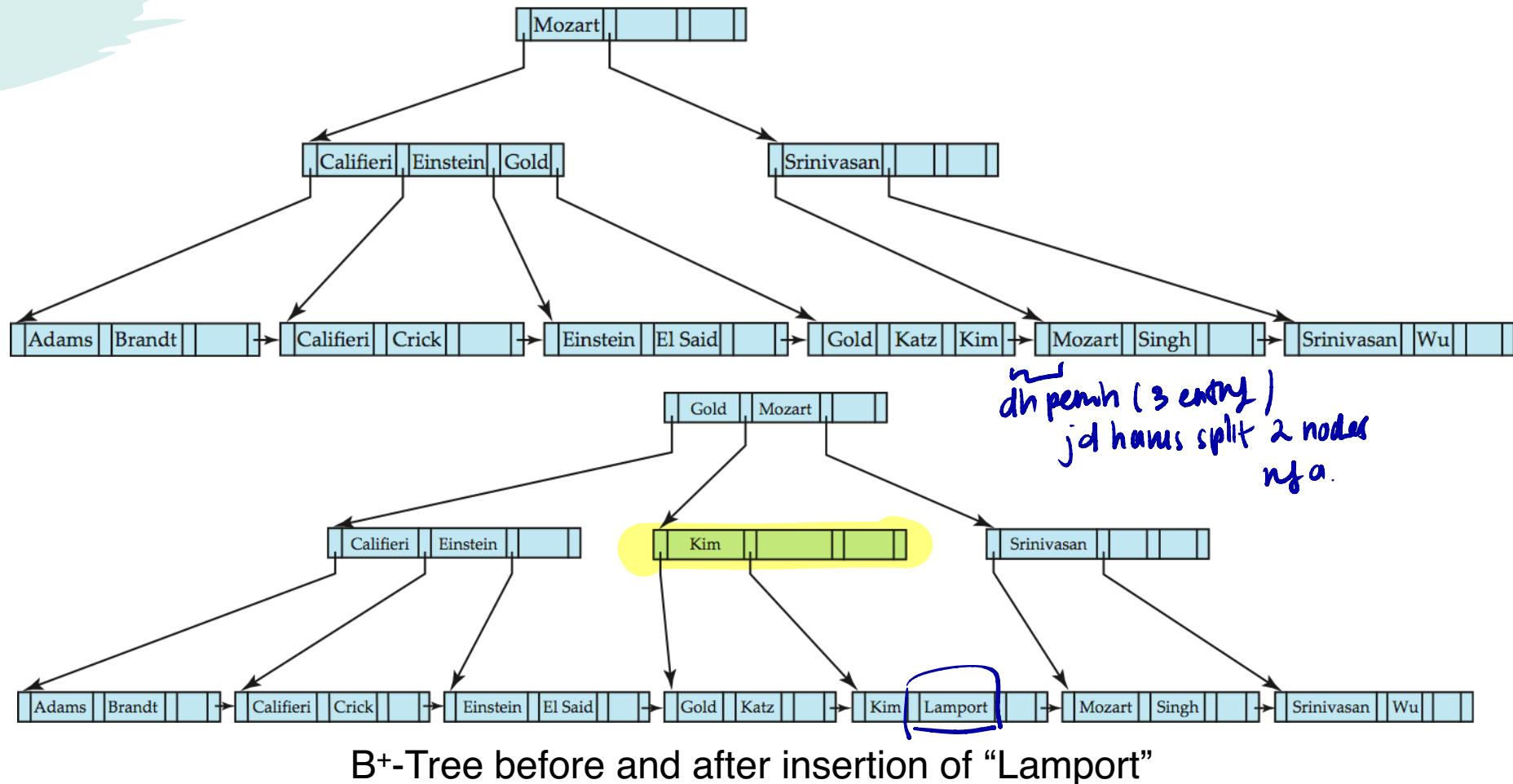
*untuk
acess
(leaf, → root, internal nodes 2x ,
and leaf..)*

B⁺-Tree Insertion



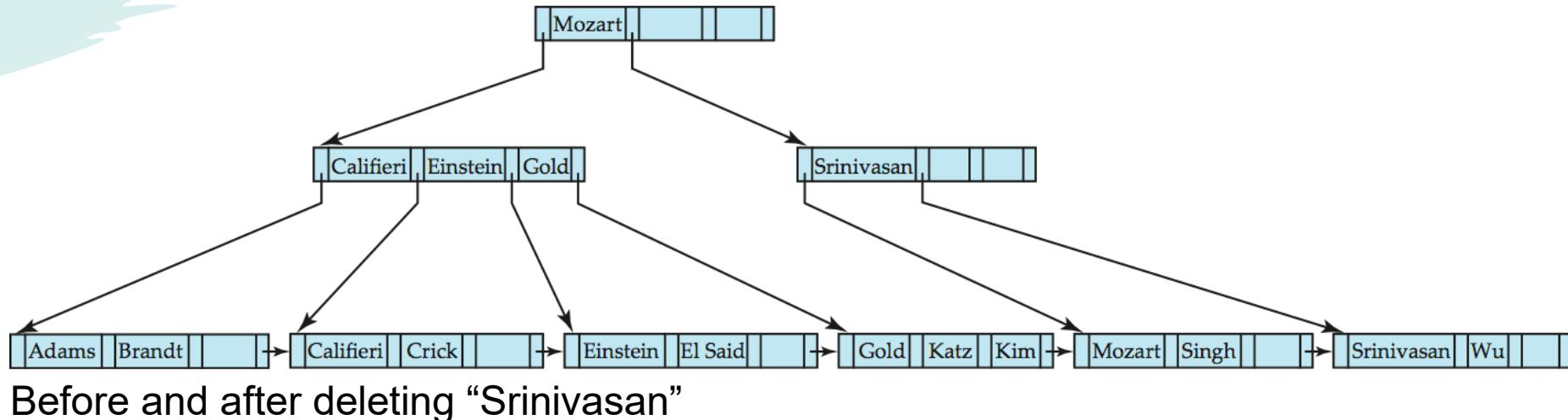
B⁺-Tree before and after insertion of “Adams”

B⁺-Tree Insertion (Cont.)

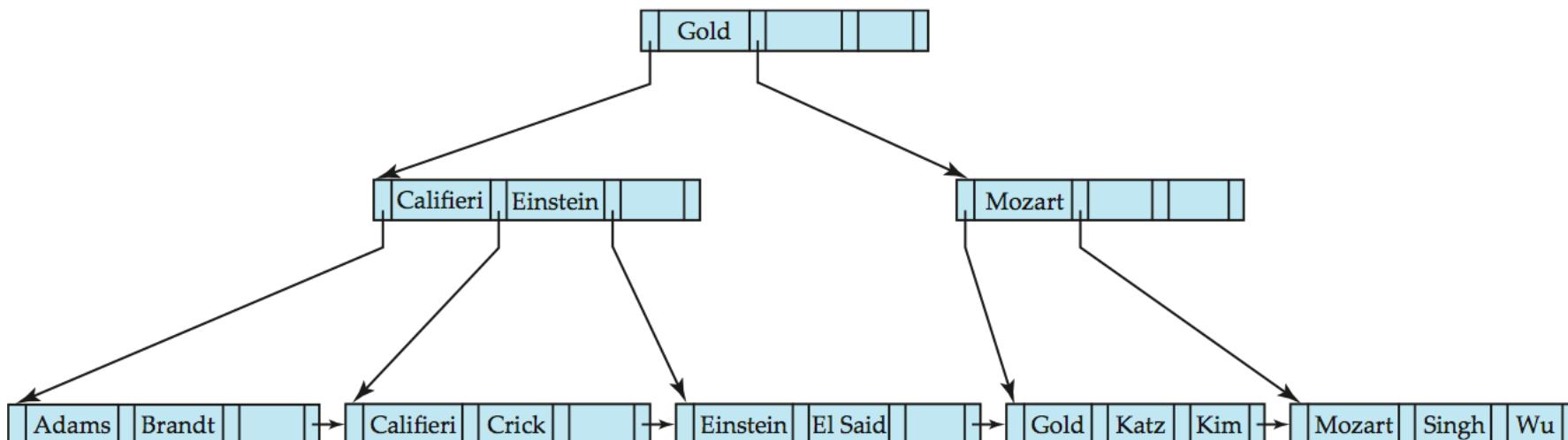


- There is **split** and **merge** mechanisms to maximize the number of search-keys in each node. Read the detail in the textbook!

Examples of B^+ -Tree Deletion

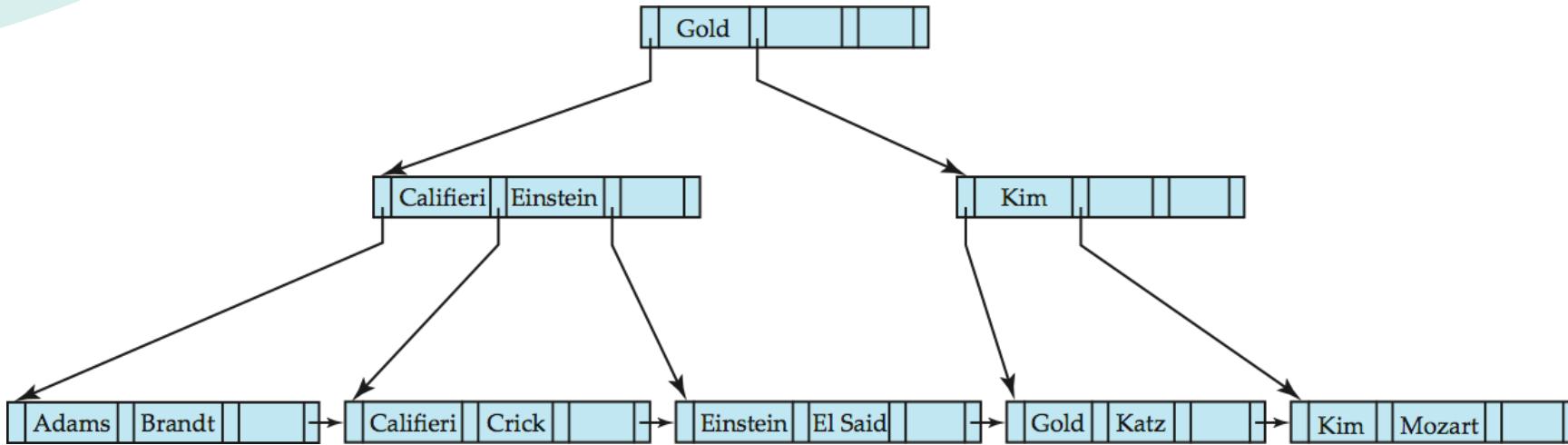


Before and after deleting “Srinivasan”



- Deleting “Srinivasan” causes merging of under-full leaves

Examples of B+-Tree Deletion (Cont.)



Deletion of “Singh” and “Wu” from result of previous example

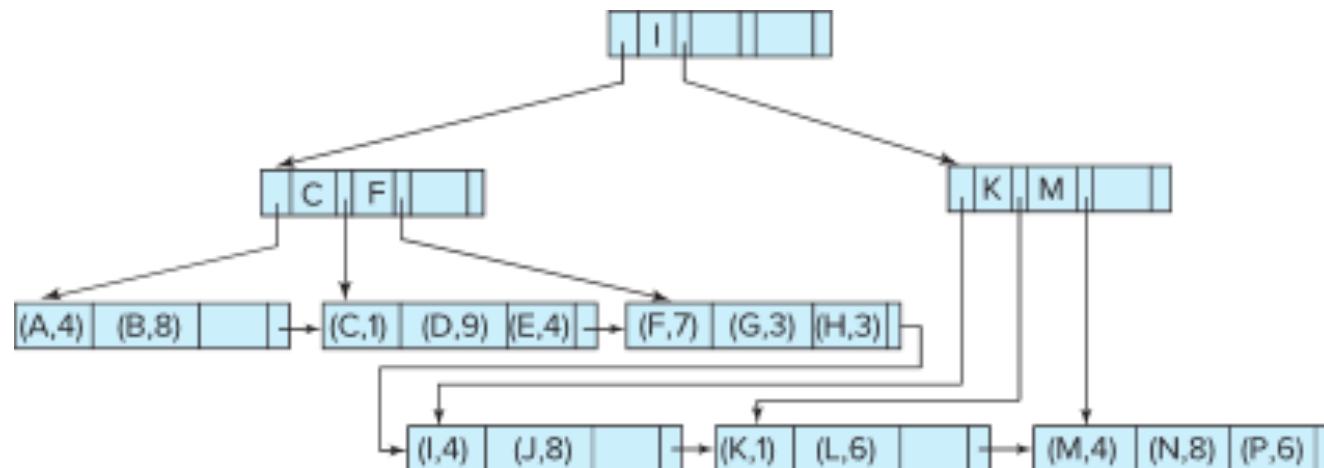
- Leaf containing Singh and Wu became underfull, and borrowed a value Kim from its left sibling
- Search-key value in the parent changes as a result

B+-Tree File Organization

- B+-Tree File Organization:
 - Leaf nodes in a B+-tree file organization store records, instead of pointers
 - Helps keep data records clustered even when there are insertions/deletions/updates
- Leaf nodes are still required to be half full
 - Since records are larger than pointers, the maximum number of records that can be stored in a leaf node is less than the number of pointers in a nonleaf node.
- Insertion and deletion are handled in the same way as insertion and deletion of entries in a B+-tree index.

B+-Tree File Organization (Cont.)

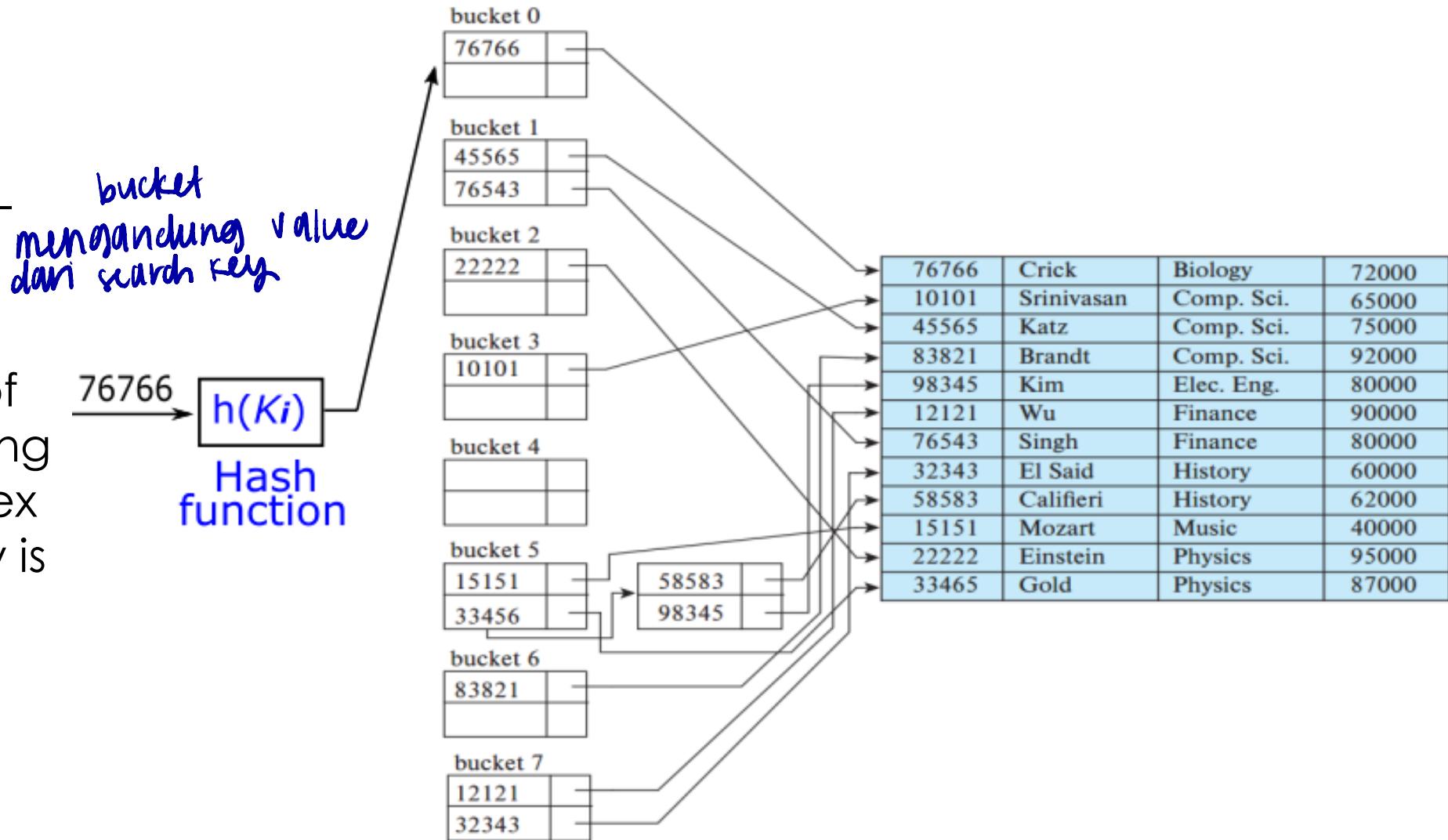
- Example of B+-tree File Organization



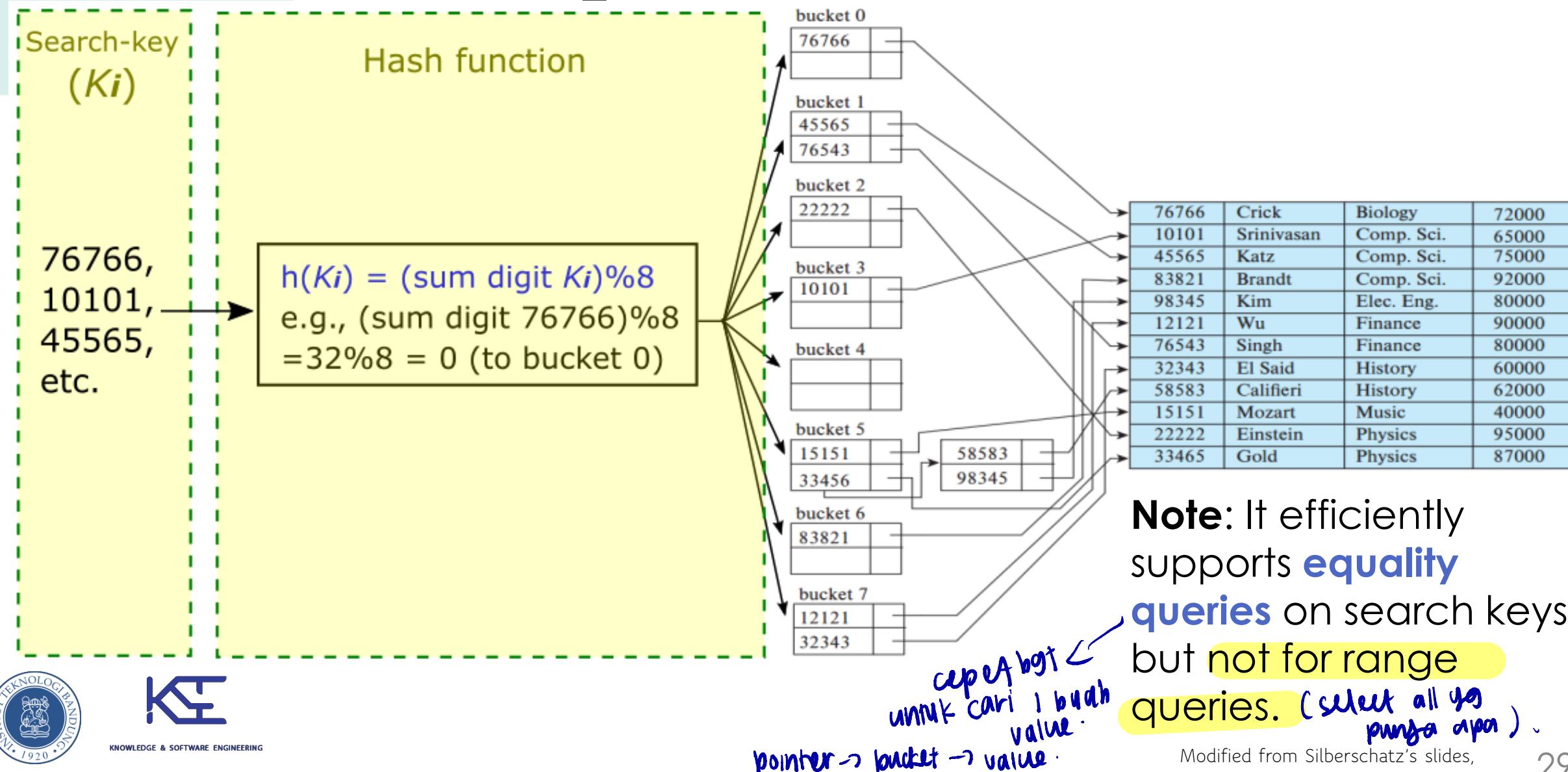
- Good space utilization important since records use more space than pointers.
- To improve space utilization, involve more sibling nodes in redistribution during splits and merges
 - Involving 2 siblings in redistribution (to avoid split / merge where possible) results in each node having at least $\lfloor 2n/3 \rfloor$ entries

Hash Index

- Hash index uses function $h(k_i)$ mapping search key value K_i to bucket.
- **Bucket** is a unit of storage containing one or more index records, typically is one block
- Bucket **contains search-key and pointer to record**

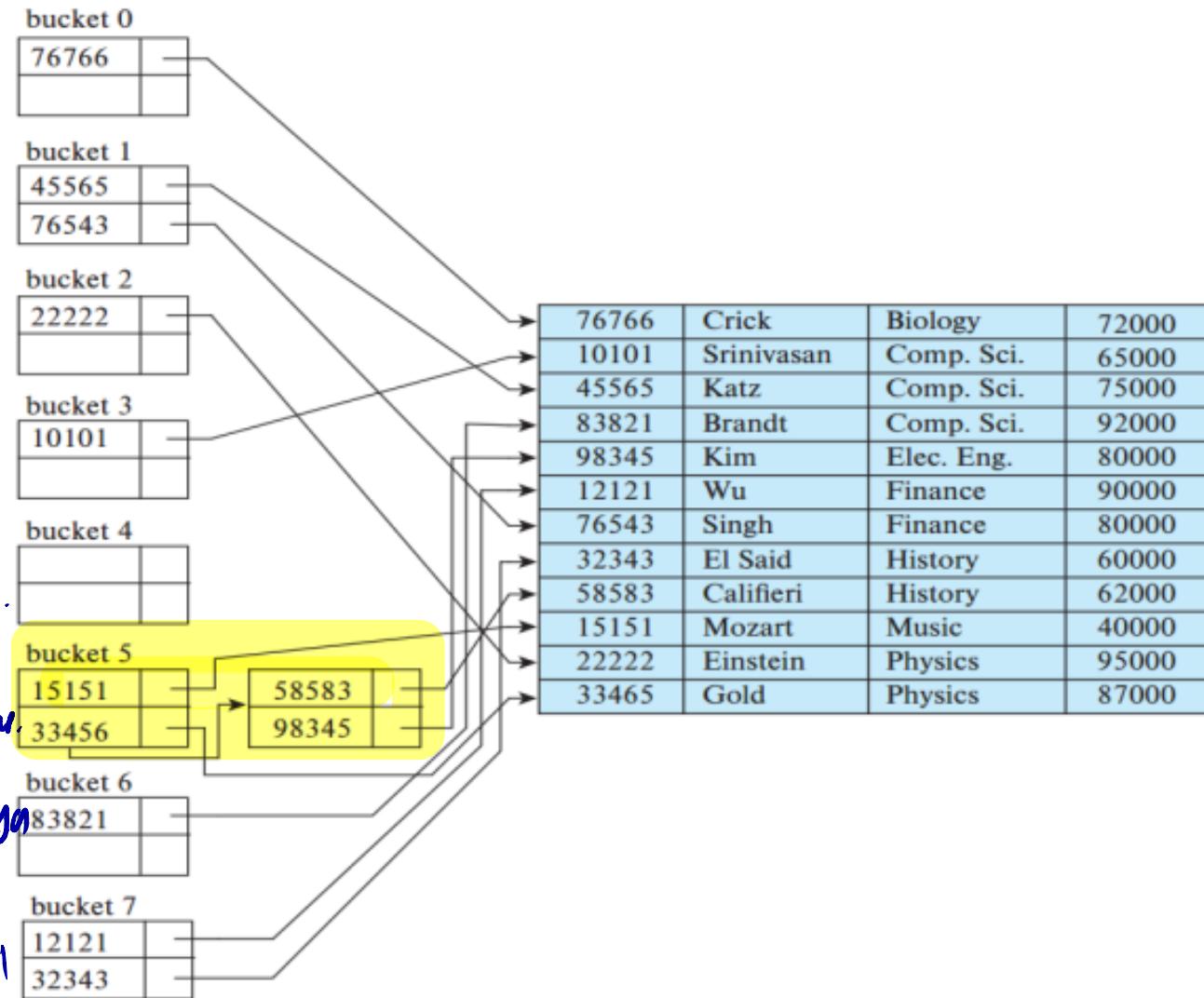


Hash Index Example



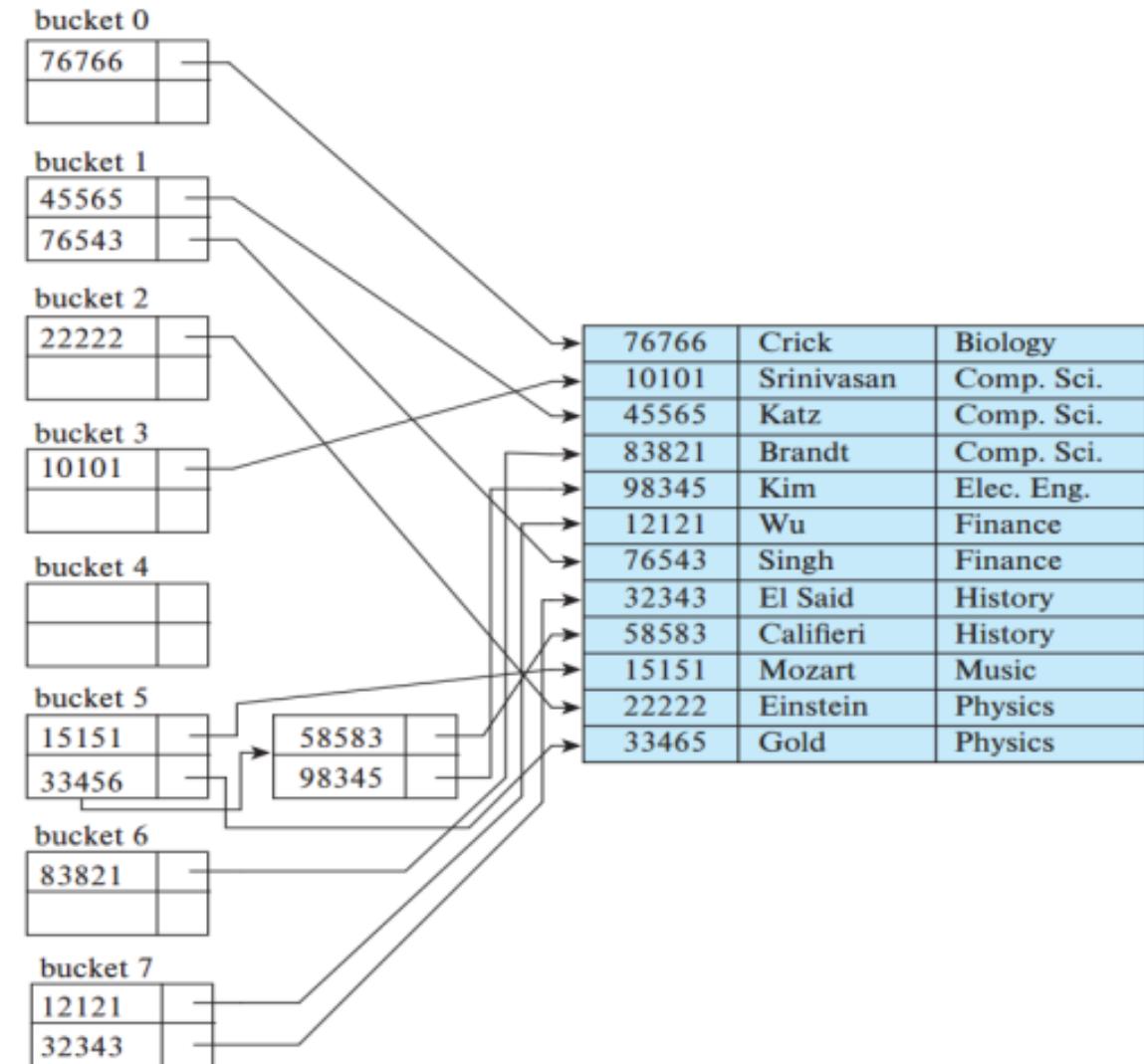
Hash Index (Cont.)

- Records with **different search-key** values may be **mapped to the same** bucket causing **overflow** bucket (see bucket 5)
- Overflow chaining** – the overflow buckets of a given bucket are chained together in a linked list.
*too many records mapped ke...
bucket dan bucket nya penuh,
butuh bucket bawah*
- In such case, entire bucket has to be searched sequentially to locate a record.



Hash Index (Cont.)

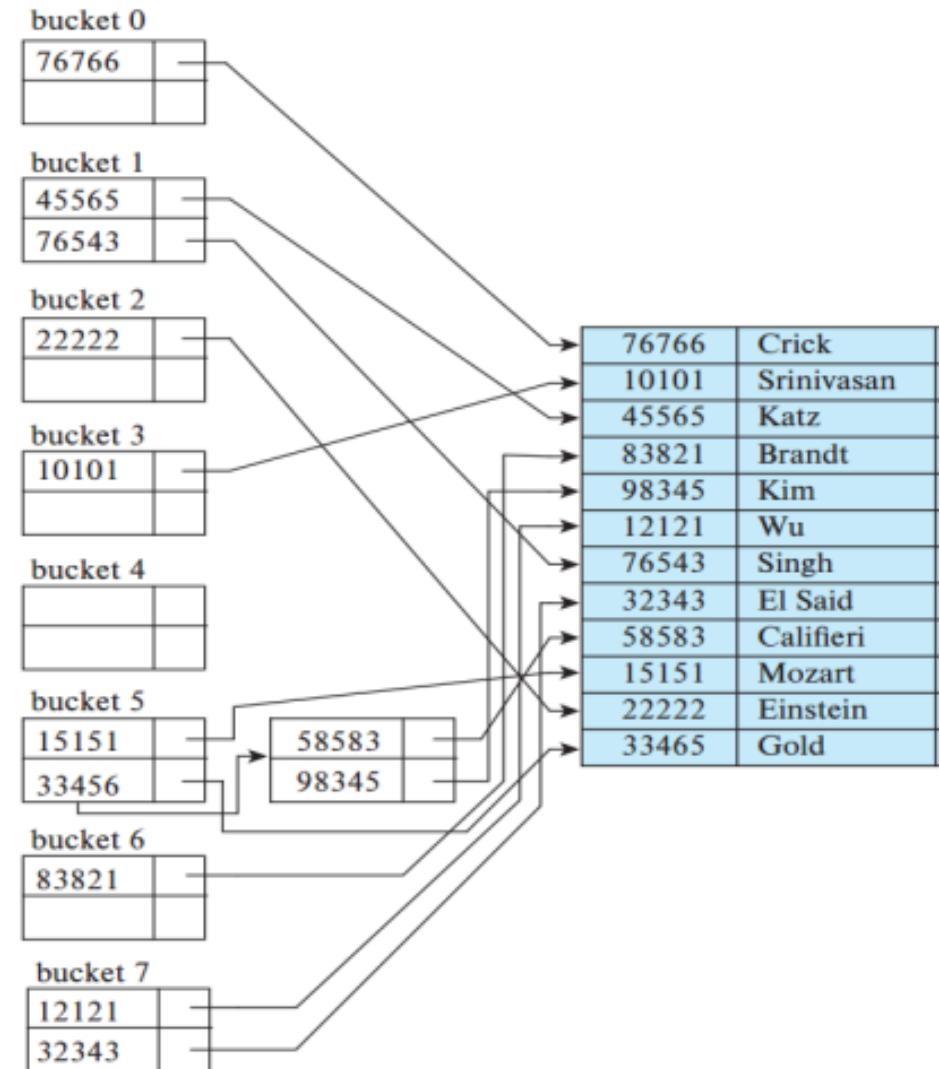
- Ideal hash function has these properties to minimize overflow bucket
 - **Uniform:** the hash function assigns each bucket the **same number** of index records
 - **Random:** each bucket will have nearly the same number of index records, **regardless of the actual distribution** of search-key values



Hash Index (Cont.)

- Type of hashing methods:
 - **Static hashing:** the set of buckets is fixed at the time the index is created → need to know how many records in advance
 - **Dynamic hashing:** the hash index can be rebuilt with an increased number of buckets

More details, read in the book



Hash File Organization

- A **bucket** is a unit of storage containing one or more records (a bucket is typically a disk block).
- In a **hash file organization** we obtain the bucket of a record directly from its search-key value using a **hash function**.
- Hash function h is a function from the set of all search-key values K to the set of all bucket addresses B .
- Hash function is used to locate records for access, insertion as well as deletion.
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate a record.

Example of Hash File Organization

Hash file organization of *instructor* file, using *dept_name* as key
(See figure in next slide.)

- There are 10 buckets,
- The binary representation of the i th character is assumed to be the integer i .
- The hash function returns the sum of the binary representations of the characters modulo 10
 - E.g. $h(\text{Music}) = 1 \quad h(\text{History}) = 2$
 $h(\text{Physics}) = 3 \quad h(\text{Elec. Eng.}) = 3$

Example of Hash File Organization (cont.)

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Hash file organization of *instructor* file, using *dept_name* as key (see previous slide for details).



Bitmap Indices

→ COOK untuk
yg varietynya
dikit

- Special type of index designed for **efficient querying on multiple keys**
- Applicable on attributes having relatively **small number of distinct values**
 - E.g., gender, country, state, etc.
 - E.g., income-level (broken up into a small number of levels, such as 0-9999, 10000-19999, 20000-50000, 50000-infinity)

record number	<i>ID</i>	<i>gender</i>	<i>income_level</i>
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

	Bitmaps for <i>gender</i>	Bitmaps for <i>income_level</i>
m	10010	L1
f	01101	L2
		L3
		L4
		L5



Bitmap Indices (Cont.)

- Bitmap indices are useful for queries on **multiple attributes**
 - not particularly useful for single attribute queries
- Queries are answered using bitmap operations
 - Intersection (and)
 - Union (or)
 - Complementation (not)
 - **E.g.**, query for **males** with income **level L1**: 10010 AND 10100 = 10000
 - Can then retrieve required tuples.
 - Counting number of matching tuples is even faster



KNOWLEDGE & SOFTWARE ENGINEERING

Index Definition in SQL

- Create an index:

create index <index-name> **on** <relation-name>(<attribute-list>)

E.g.: **create index** b-index **on** branch(branch_name)

- Use **create unique index** to indirectly specify and enforce the condition that the search key is a candidate key.
 - Not really required if SQL **unique** integrity constraint is supported
- To drop an index

drop index <index-name>

- Most database systems allow specification of type of index, and clustering.

End of Topic



KNOWLEDGE & SOFTWARE ENGINEERING



Modified from Silberschatz's slides,
"Database System Concepts", 7th ed.

IF2240 – Basis Data Schema and Index Tuning

SEMESTER II TAHUN AJARAN 2022/2023



KNOWLEDGE & SOFTWARE ENGINEERING

Modified from [Silberschatz's slides](#),
“Database System Concepts”, 7th ed.



Sources

- “Database Systems: Design, Implementation, and Management”, ninth edition, by Coronel, Morris, Rob; chapter 11 (Cengage Learning, 2010)
- “Database System Concepts”, seventh edition, Silberschatz, Korth, Sudarshan, chapter 25 (McGraw Hill, 2019)
- Other sources

Objectives

- Students are able to perform higher level database design tuning (i.e. schema tuning, materialized view, and index tuning) in order to improve the database performance

Higher-Level Database Design Tuning

- After ER design, schema refinement, and the definition of views, we have the **conceptual** and **external** schemas for our database.
- The next step is to choose **indexes** and to **refine** the conceptual and external schemas (if necessary) to meet performance goals.
- We must begin by understanding the **workload**:
 - The most important queries and how often they arise.
 - The most important updates and how often they arise.
 - The desired performance for these queries and updates.

Understanding the Workload

- For each query in the workload:
 - Which relations does it access?
 - Which attributes are retrieved?
 - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- For each update in the workload:
 - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
 - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

Decisions to Make

- What indexes should we create?
 - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
 - Clustered/non-clustered? Hash/B⁺-tree? Bitmap?
- Should we make changes to the logical schema?
 - Consider alternative normalized schemas? (Remember, there are many choices in decomposing into BCNF, etc.)
 - Should we “undo” some decomposition steps and settle for a lower normal form? (Denormalization)
 - Horizontal partitioning, replication, views,



Schema Tuning



KNOWLEDGE & SOFTWARE ENGINEERING

Schema Tuning (I)

- Choice of logical schema should be guided by workload, in addition to redundancy issues:
 - We might consider **horizontal decompositions**.
 - We might consider **vertical decompositions**.
 - We may settle for a 3NF schema rather than BCNF.
 - Workload may influence choice we make in decomposing a relation into 3NF or BCNF.
 - We may further decompose a BCNF schema!
 - We might **denormalize** (i.e., undo a decomposition step), or we might add fields to a relation.
- If such changes are made after a database in use, called **schema evolution**; might mask changes by defining views

soalnya BCNF
lebih cluttered.

Schema Tuning (2)

- Can be done in several ways:

- **Splitting tables**

- Sometimes splitting normalized tables can improve performance
- Can split tables in two ways: (1) **Horizontally**; (2) **Vertically** → menambah kompleksitas
- Add complexity to the applications

- **Denormalization**

- Adding redundant columns
- Adding derived attributes
- Collapsing tables
- Duplicating tables

Schema Tuning: Horizontal Splitting (I)

or more
(depends)

- **Horizontal splitting:** Placing rows in two separate tables, depending on data values in one or more columns
- Use horizontal splitting if:
 - A table is large, and reducing its size reduces the number of index pages read in a query
 - The table split corresponds to a natural separation of the rows, such as different geographical sites or historical vs current data
 - Might choose horizontal splitting if a table stores huge amounts of rarely used historical data, and the applications have high performance needs for current data in the same table
 - Table splitting distributes data over the physical media

if u have > 1 server,
jd datanya di split + terus
di pisah².

Schema Tuning: Horizontal Splitting (2)

Horizontal Splitting Example

Problem:

Usually only active records are processed

Authors		
Active		
Active		
Inactive		
Active		
Inactive		
Inactive		

Solution: Partition horizontally into active and inactive data

Active_Authors			Inactive_Authors		
Active			Inactive		
Active			Inactive		
Active			Inactive		

still same schema!, jangan hapus!

Schema Tuning: Vertical Splitting (I)

- **Vertical Splitting:** Partition relations to isolate the data that is accessed most often – only fetch needed information
- Keeps the relation in normal form
- Use vertical splitting if:
 - Some columns are accessed more frequently than other columns
 - The table has wide rows, and splitting the table reduces the number of pages that need to be read
- Makes even more sense when both of the above conditions are true
 - When a table contains very long columns that are accessed infrequently, placing them in a separate table can greatly speed the retrieval of the more frequently used columns
 - With shorter rows, more data rows fit on a data page, so for many queries, fewer pages can be accessed

Schema Tuning: Vertical Splitting (2)

Vertical Splitting Example

- Account relation with the following schema:

account (account-number, branch-name, balance)

- Can be split into two relations:

account-branch (account-number, branch-name)
account-balance (account-number, balance)

- The two representations are equivalent and in the maximum normal form (BCNF). But they could give different performance characteristics, depends on the information that usually retrieved at the same time. E.g. branch-name need not be fetched unless required.

account-number
di account-branch
harus sama dgn
 yg di account-
balance
[foreign-key
reference]

Schema Tuning: Denormalization (I)

untuk melakukan kalkulasi complex (cth: IP)

- Can be done with tables or columns
- Assumes prior normalization
- Requires a thorough knowledge of how the data is being used
- Used if:
 - All or nearly all of the most frequent queries require access to the full set of joined data
 - A majority of applications perform table scans when joining tables
 - Computational complexity of derived columns requires temporary tables or excessively complex queries

banyak menyebabkan redundansi.
yg bisa
dianalisis untuk
di tackle

kalau data
sering dilihat bareng,
boleh dipertimbangkan
untuk denormalisasi

Schema Tuning: Denormalization (2)

- Denormalization can improve performance by:
 - Minimizing the need for joins
 - Reducing the number of foreign keys on tables
 - Reducing the number of indices, saving storage space, and reducing data modification time
 - Precomputing aggregate values
 - Reducing the number of tables (in some cases)
- Disadvantages:
 - It usually speeds retrieval but can slow data modification
 - It is always application-specific and must be re-evaluated if the application changes
 - It can increase the size of tables
 - In some instances, it simplifies coding; in others, it makes coding more complex

Schema Tuning: Denormalization (3)

- Issues to examine when considering denormalization include:
 - What are the critical transactions, and what is the expected response time
 - How often are the transactions executed?
 - What tables or columns do the critical transactions use? How many rows do they access each time?
 - What is the mix of transaction types: select, insert, update, and delete
 - What is the usual sort order?
 - What are the concurrency expectations?
 - How big are the most frequently accessed tables?
 - Do any processes compute summaries?
 - Where is the data physically located?

Schema Tuning: Denormalization

Adding Redundant Columns

- To eliminate frequent joins for many queries
- For example: if performing frequent joins on the **titleauthor** and **authors** tables to retrieve author's last name, you can add the **au_Iname** column to **titleauthor**
- The problems with this solution are that it:
 - Requires maintenance of new columns
 - Have to make changes to two tables, and possibly to many rows in one of the tables
 - Requires more disk space

→ basically
menyulikat

common query.

```
select title, au_Iname  
from titleauthor ta, authors a  
where ta.au_id = a.au_id
```

titleauthor		authors	
title	title_id	au_id	au_Iname

foreign key ref

join columns

jadi gausah join

```
select title, au_Iname from titleauthor
```

titleauthor		authors			
title_id	au_id	au_Iname	advance	au_id	au_Iname

problem: kalau nanti ada perubahan di table authors, di titleauthor juga harus berubah . ingat: delete, cascade, set null

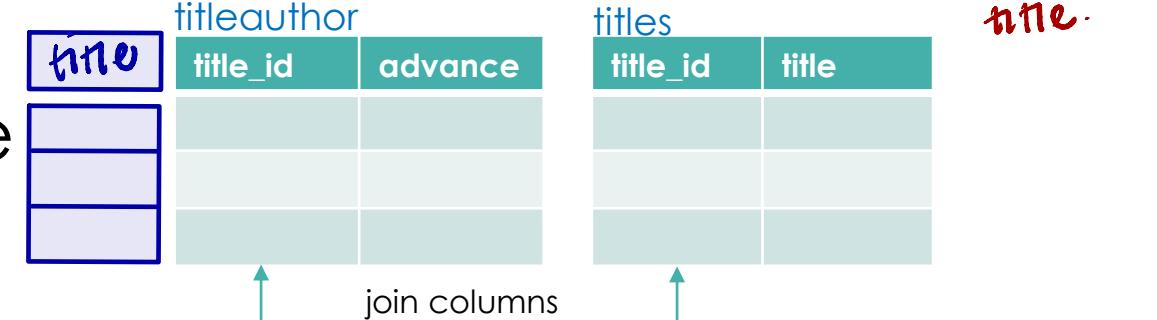
Schema Tuning: Denormalization

Adding Derived Columns

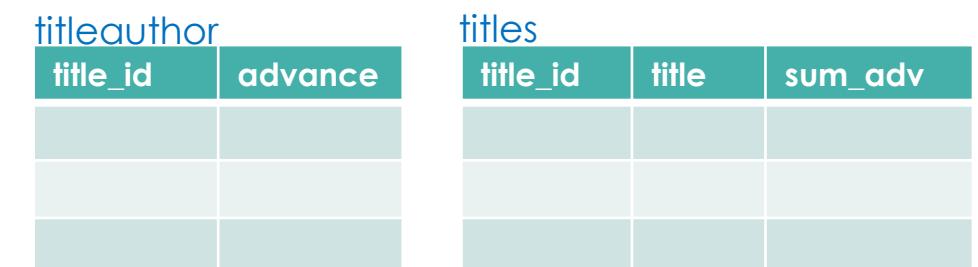
- Can eliminate some joins and reduce the time needed to produce **aggregate values**
- E.g.: The **sum_adv** column in the **titles** table
 - Eliminating both the join and the aggregate at runtime
 - Increases storage needs and requires maintenance of the derived column wherever changes are made to the titles table

```
select title, sum(advance)  
from titleauthor ta, titles t  
where ta.title_id = t.title_id  
group by title
```

mau menghitung
sum advance
dari seiap
title.



```
select title_id, sum_adv from titles
```



Schema Tuning: Denormalization

Collapsing Tables

- If most users need to see the full set of joined data from two tables, collapsing the two tables into one can improve performance by eliminating the join
- The data from two tables must be in a **one-to-one** relationship to collapse tables
- Eliminates the join, but loses the conceptual separation of the data
 - If some users still need access to just the pairs of data from the two tables, this access can be restored by using queries that select only the needed columns or by using views

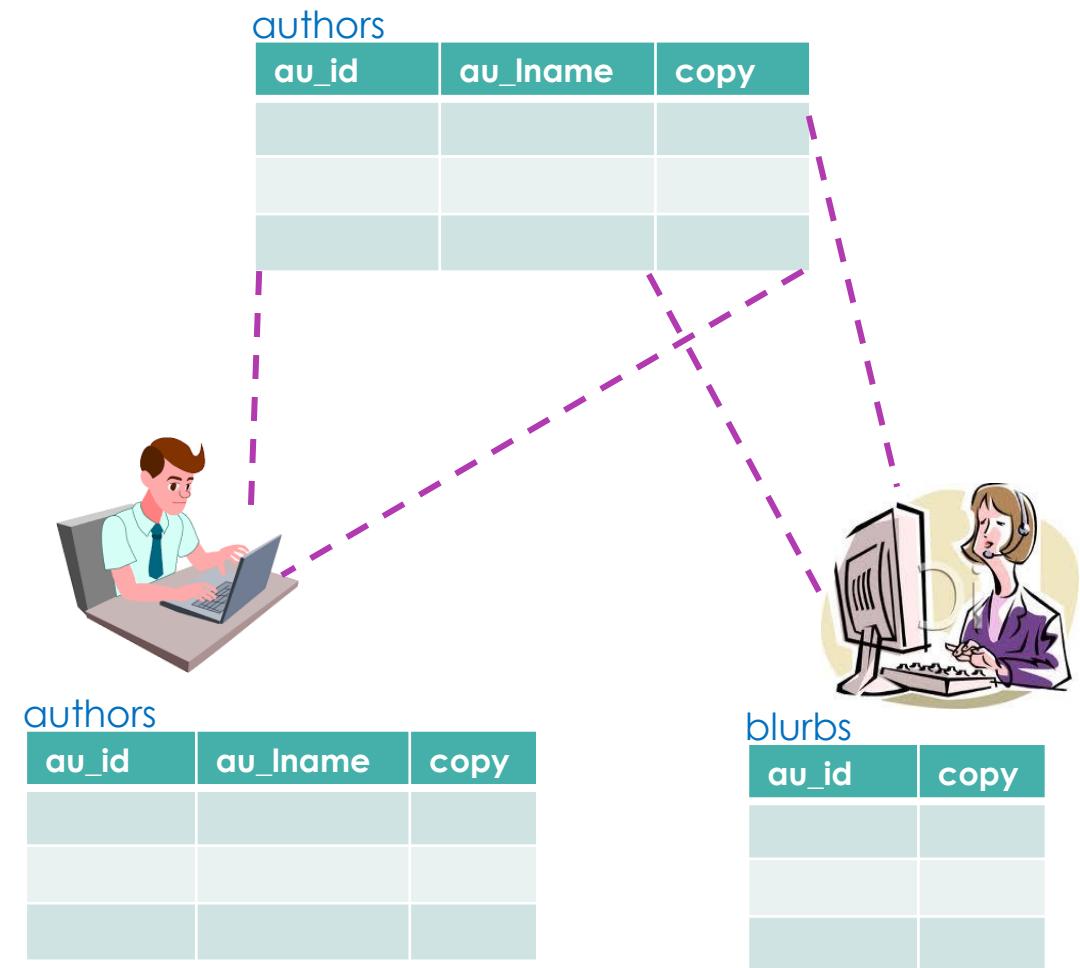
contoh: data penduduk dan
data mahasiswa
mahasiswa sbrnnya penduduk.
jd digabung? tups!
jd ga tau yg mana
data kependudukan, yg mana data mahasiswa

Schema Tuning: Denormalization

Duplicating Tables

- If a group of users regularly needs only a subset of data, the critical table subset can be duplicated for that group
- Minimizes contention, but requires managing redundancy

another way of doing this : create view



Schema Tuning: Managing Denormalized Data

- Need to ensure data integrity by using:
 - **Triggers**, which can update derived or duplicated data anytime the base data changes → **mechanism when u do update / insert / delete**.
 - **Application logic**, using transactions in each application that update denormalized data, to ensure that changes are atomic
 - Be very sure that the data integrity requirements are well documented and well known to all application developers and to those who must maintain applications
 - **Batch reconciliation**, run at appropriate intervals, to bring the denormalized data back into agreement
 - If 100 percent consistency is not required at all times
- From an integrity point of view, triggers provide the best solution, although they can be costly in terms of performance

misal : tipe x ada perubahan au_lname di table author, nanti akan trigger perubahan otomatis di au_lname di titleauthor jd consisten terus

titleauthor	au_id	au_lname	advance	authors	au_id	au_lname
1	1	Author A	0	2	1	Author A
2	2	Author B	1	3	2	Author B

Tuning the Database Design: Materialized view (I)

- **Materialized views** can help speed up certain queries
 - Particularly aggregate queries
- Overheads
 - Space
 - Time for view maintenance
 - Immediate view maintenance: done as part of update transactions
 - time overhead paid by update transaction
 - Deferred view maintenance: done only when required
 - update transaction is not affected, but system time is spent on view maintenance
 - until updated, the view may be out-of-date
- Preferable to denormalized schema since view maintenance is systems responsibility, not programmers
 - Avoids inconsistencies caused by errors in update programs

Tuning the Database Design: Materialized view (2)

↳ datanya beneran disimpan @ harddisk

- How to choose set of materialized views
 - Helping one transaction type by introducing a materialized view may hurt others
 - Choice of materialized views depends on costs
 - Users often have no idea of actual cost of operations
 - Overall, manual selection of materialized views is tedious
- Some database systems provide tools to help DBA choose views to materialize
 - “Materialized view selection wizards”



Index Tuning



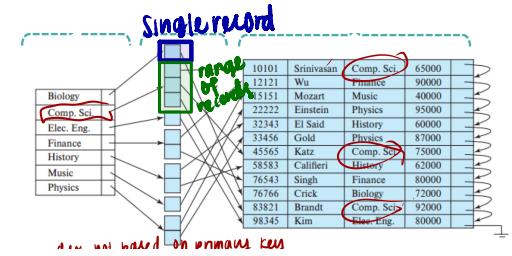
KNOWLEDGE & SOFTWARE ENGINEERING

bisa ditambah di primary key tapi ga selalu.

Primary (Clustering) Indices

- Primary (clustering) indices determines the order of physical records.
 - A **SELECT** statement with no **ORDER BY** clause will return data in the clustering index order.
 - 1 clustered index per table, 249 non-clustered indexes per table.
- Highly recommended for every table!
- Very useful for **columns** sorted on **GROUP BY** and **ORDER BY** clauses, as well as those filtered by WHERE clauses.

Secondary (Non-Clustering) Indices



- Useful for retrieving a **single record** or a **range of records**.
- Maintained in a separate structure and maintained as changes are made to the base table.
- Tend to be **much smaller** than the base table, so they can locate the exact record(s) with much less I/O.
- Any time you **rebuild** the primary (clustering) index, you **also automatically rebuild** all secondary (non-clustering) index on the table.



KNOWLEDGE & SOFTWARE ENGINEERING

Index Tuning

→ nge-tun behaviour querynya dulu.

- Create **appropriate indices to speed up** slow queries/updates.
- Speed up **slow updates by removing** excess indices (tradeoff between queries and updates). → removing indeks yg emang ga dipake.
→ banyak operasi == di query.
- **Choose type of index** (B-tree/**hash**/Bitmap) appropriate for most frequent types of queries.
bagus kalau datanya punya very few distinct value
- Choose **which index** to be made **clustered**.
- **Index tuning wizards** look at past history of queries and updates (the **workload**) and **recommend** which indices would be best for the workload.



KNOWLEDGE & SOFTWARE ENGINEERING

When to use each type of index? (I)

- Primary (clustered) indices are actual physically written records → Highly recommended for every table
 - Very useful for columns sorted in GROUP BY or ORDER BY as well as those filtered by WHERE clause
- Secondary (non-clustered) indices are useful for retrieving a single or a range of records
- B⁺-tree
 - It takes approximately the same amount of time to access any given row in the index
 - The default and most common type of index used in DBMS
 - Used mainly in columns with values repeat in a relatively small number → high data sparsity

When to use each type of index? (2)

- Hash index:
 - Good for simple and fast lookup operations based on equality conditions, e.g. LNAME = 'Scott'; FNAME = 'Shannon'
- Bitmap index: → *analitical application, banyak query read.*
 - Used mostly in data warehouse applications in tables with large number of rows in which a small number of column values repeat many times → low data sparsity



KNOWLEDGE & SOFTWARE ENGINEERING

When should index be considered?

- **Unique index** are implicitly used in conjunction with a **primary key** for the primary key to work (dbms typically create index on primary-key)
- **Foreign keys** are also excellent candidates for an index because they are often used to join the parent table.
 - Most, if not all, columns used for table joins should be indexed.
- Columns that are frequently referenced in the **ORDER BY** and **GROUP BY** clauses should be considered for being indexed.
- **MAX** or **MIN** function is applied to index column

L) cepat karena tinggal
ambil awal dan
akhir record .

(dia dah
ordered)



KNOWLEDGE & SOFTWARE ENGINEERING

When should indexes be considered? (Cont.)

- Index should be created on columns with a **high number of unique values**, of columns when used as filter conditions in the **WHERE** clause **return a low percentage** of rows of data from a table
- The effective use of indexes requires a thorough knowledge of table relationships, query and transaction requirements, and the data itself.



KNOWLEDGE & SOFTWARE ENGINEERING

Add an index when

- Data within the column is **frequently accessed**, particularly if it is used to **filter** data either by **specific value** or by **range** of values
 - For example:
 - **WHERE emp_SSN = '123-56-7890'**
 - **WHERE emp_hire_date BETWEEN 'Jan 01, 2000' AND 'Nov 01, 2002'**
- Data within the column is **used to build joins**. Foreign keys are almost always good candidates for being indexed.
 - For example:

```
SELECT DISTINCT b.title
FROM book AS b
JOIN sales AS s ON b.title_id = s.title_id
```
 - **title_ID** on the **sales** table should be indexed

Index Tuning (cont.)

When should indexes be avoided?

- Indexes should not be used **on small tables**.
- Indexes should not be used on columns that **return a high percentage of data** rows when used as a filter condition in a query's WHERE clause.
- Tables that have frequent, large batch update jobs run can be indexed.
 - However, the batch job's performance is slowed considerably by the index.
*indexes
baus
batch job
update ny
jalan*
 - Might consider dropping the index before the batch job, and then recreating the index after the job has completed. → *mending drop index dulu, update data,
tutak lagi index*.
- Indexes should not be used on columns that contain a **high number of NULL** values.
- Columns that are **frequently manipulated** should not be indexed.
 - Maintenance on the index can become excessive.



KNOWLEDGE & SOFTWARE ENGINEERING

Index Tuning (cont.)

- **Rebuild** your index often
- Gather **statistics**
- Do not **overuse** indices
- Restrict to columns that **return a few records**
- Use **Bitmap index** when number of values is small e.g., sex: male, female, and filtering involving multi attributes
- Suppression of index

select * from mytable where total + 3 = 20

, Kadanya² sdh tarok index di kolom
tp indexnya itu ga dipakai di
querynya

caranya
mendiny
where
total = 17 .
(index
bisa
repate)



How Many Indices?

- Adding **non-clustered indices** to a table can **greatly speed SELECT statements**.
- Every index has a certain amount of overhead.
 - The greater the number of indices, the more overhead with every INSERT, UPDATE, and DELETE statement.
- Must balance the needs of the application with the pros and cons of added indices
- **DSS** (Decision Support System) vs. **OLTP** (Online Transaction Processing)



KNOWLEDGE & SOFTWARE ENGINEERING

Index Tuning Wizard

- The Index Tuning Wizard allows you to select and create an optimal set of indices and statistics **without requiring an expert** understanding of the structure of the database, the workload, or the internals of db server
 - E.g. Microsoft® SQL Server™ 2000
- Returns best results using a realistic workload captured using SQL Profiler
- Recommendations returned by the wizard are SQL commands that can be run directly against the database in SQL Query Analyzer.

IF3140 Sistem Basis Data SQL Performance Tuning

SEMESTER I TAHUN AJARAN 2024/2025



KNOWLEDGE & SOFTWARE ENGINEERING



Sources

- “Database Systems: Design, Implementation, and Management”, ninth edition, by Coronel, Morris, Rob; chapter 11 (Cengage Learning, 2010)



KNOWLEDGE & SOFTWARE ENGINEERING

SQL Performance Tuning

- Most current-generation relational DBMSs perform automatic query optimization at the server end.
- Most SQL performance optimization techniques are DBMS-specific and, therefore, are rarely portable, even across different versions of the same DBMS.
- **Does this mean that you should not worry about how a SQL query is written because the DBMS will always optimize it?**
- No, because there is considerable room for improvement.
- The DBMS uses general optimization techniques, rather than focusing on specific techniques dictated by the special circumstances of the query execution.



KNOWLEDGE & SOFTWARE ENGINEERING

SQL Performance Tuning

- A poorly written SQL query can, and *usually will*, bring the database system to its knees from a performance point of view.
- The majority of current database performance problems are related to poorly written SQL code.
- A carefully written query almost always outperforms a poorly written one.



KNOWLEDGE & SOFTWARE ENGINEERING

SQL Performance Tuning

- Most recommendations in this material are related to the use of the SELECT statement.
- Recommendation categories:
 - Index tuning (revisited)
 - Conditional expressions
 - Query formulation



KNOWLEDGE & SOFTWARE ENGINEERING

Index Tuning - Revisited

- Indexes are the most important technique used in SQL performance optimization
- General rule of when indexes are likely to be used:
 - When an indexed column appears by itself in a search criteria of a WHERE or HAVING clause.
 - When an indexed column appears by itself in a GROUP BY or ORDER BY clause.
 - When a MAX or MIN function is applied to an indexed column.
 - When the data sparsity on the indexed column is high.



KNOWLEDGE & SOFTWARE ENGINEERING

Index Tuning - Revisited

- Some general guidelines for creating and using indexes
 - Create indexes for each single attribute used in a WHERE, HAVING, ORDER BY, or GROUP BY clause
 - Do not use indexes in small tables or tables with low sparsity.
 - Declare primary and foreign keys so the optimizer can use the indexes in join operations.
 - Declare indexes in join columns other than PK or FK



KNOWLEDGE & SOFTWARE ENGINEERING

Index Tuning – Revisited

- You cannot always use an index to improve performance:
 - In some DBMSs, indexes are ignored when you use **functions** in the table attributes
 - But major databases (such as Oracle, SQL Server, and DB2) now support function-based indexes
 - A **function-based index** is an index based on a specific SQL function or expression. Function-based indexes are especially useful when dealing with derived attributes.
 - Example: an index on YEAR(INV_DATE).
 - Too many indexes will slow down INSERT, UPDATE, and DELETE operations
 - Some query optimizers will choose only one index to be the driving index for a query, even if your query uses conditions in many different indexed columns

Conditional Expressions

- A conditional expression is normally placed within the WHERE or HAVING clauses of a SQL statement
- A conditional expression restricts the output of a query to only the rows matching the conditional criteria
- Most of the query optimization techniques mentioned next are designed to make the optimizer's work easier.



KNOWLEDGE & SOFTWARE ENGINEERING

Conditional Expressions

- Examples:

OPERAND1	CONDITIONAL OPERATOR	OPERAND2
P_PRICE	>	10.00
V_STATE	=	FL
V_CONTACT	LIKE	Smith%
P_QOH	>	P_MIN * 1.10

- An operand can be:
 - A simple column name such as P_PRICE or V_STATE.
 - A literal or a constant such as the value 10.00 or the text 'FL'.
 - An expression such as P_MIN * 1.10.



KNOWLEDGE & SOFTWARE ENGINEERING

Conditional Expressions

Some **common practices** used to write efficient conditional expressions:

- Use *simple columns or literals* as operands in a conditional expression—avoid the use of conditional expressions with functions whenever possible.

Examples:

- $P_PRICE > 10.00$ is faster than $P_QOH > P_MIN * 1.10$
 - because the DBMS must evaluate the $P_MIN * 1.10$ expression first.
- $UPPER(V_NAME) = 'JIM'$ is slower $V_NAME = 'Jim'$ if all names in the V_NAME column are stored with proper capitalization

Conditional Expressions

- Numeric field comparisons are faster than character, date, and NULL comparisons.
 - CPU handles numeric comparisons (integer and decimal) faster than character and date comparisons
 - Indexes do not store references to null values
 - involve additional processing
 - Tend to be the **slowest** of all conditional operands.



KNOWLEDGE & SOFTWARE ENGINEERING

Conditional Expressions

- Equality comparisons are faster than inequality comparisons
 - Example:
 - P_PRICE = 10.00 is processed faster → a direct search using the index in the column.
 - An inequality symbol (>, >=, <, <=) → There will almost always be more “greater than” or “less than” values than exactly “equal” values in the index → need additional processing to complete the request
 - The slowest of all comparison operators is LIKE with wildcard symbols, as in V_CONTACT LIKE "%glo%"
 - With the exception of null (see previous slide)
 - “not equal” symbol (<>) yields slower searches, especially when the sparsity of the data is high



KNOWLEDGE & SOFTWARE ENGINEERING

Conditional Expressions

- Whenever possible, transform conditional expressions to use literals
 - Examples:
 - change $P_PRICE - 10 = 7$ to $P_PRICE = 17$
- When using multiple conditional expressions, write the equality conditions first.
 - Remember: Equality comparisons are faster than inequality comparisons
 - Examples:
 - change $P_QOH < P_MIN \text{ AND } P_MIN = P_REORDER \text{ AND } P_QOH = 10$ to $P_QOH = 10 \text{ AND } P_MIN = P_REORDER \text{ AND } P_MIN > 10$

Conditional Expressions

- If you use multiple AND conditions, write the condition most likely to be false first
 - DBMS will stop evaluating the rest of the conditions as soon as it finds a conditional expression that is evaluated as false.
 - Example:
 - $P_PRICE > 10 \text{ AND } V_STATE = 'FL'$
 - If you know that only a few vendors are located in Florida, you could rewrite the condition as: $V_STATE = 'FL' \text{ AND } P_PRICE > 10$
- When using multiple OR conditions, put the condition most likely to be true first.
 - DBMS will stop evaluating the remaining conditions as soon as it finds a conditional expression that is evaluated as true.

Conditional Expressions

- Whenever possible, try to avoid the use of the NOT logical operator.
 - Example:
 - NOT (P_PRICE > 10.00) can be written as P_PRICE <= 10.00
 - NOT (EMP_SEX = 'M') can be written as EMP_SEX = 'F'



KNOWLEDGE & SOFTWARE ENGINEERING

Query Formulation

- To formulate a query, you would normally follow the steps outlined below:
 - *Identify what columns and computations are required:*
 1. What columns do you need? All columns, or only certain columns?
 - If you need only certain columns don't use select *, since the size of data to be transferred will be greater
 2. Do you need simple expressions? Do you need functions? Do you need aggregate functions?
 3. Determine the granularity of the raw data required for your output.
 - You might need to summarize data not readily available on any table
 - You might consider breaking the query into multiple subqueries and storing those subqueries as views



KNOWLEDGE & SOFTWARE ENGINEERING

Query Formulation

- Identify the source tables
 - Try to use the least number of tables in your query to minimize the number of join operations
- Determine how to join the tables
 - Properly identify what and how to join the tables
 - In some cases, you will need some type of natural join, but others, you might need to use an outer join



KNOWLEDGE & SOFTWARE ENGINEERING

Query Formulation

- Determine what selection criteria is needed.
 - Simple comparison. For example, P_PRICE > 10.
 - Single value to multiple values. For example, V_STATE IN ('FL', 'TN', 'GA').
 - Nested comparisons → nested selection criteria involving subqueries.
For example: P_PRICE >= (SELECT AVG(P_PRICE) FROM PRODUCT).
 - Grouped data selection → the HAVING clause
- Determine *in what order to display the output*.
 - you may need to use the ORDER BY clause → ORDER BY clause is one of the most resource-intensive operations for the DBMS.



KNOWLEDGE & SOFTWARE ENGINEERING

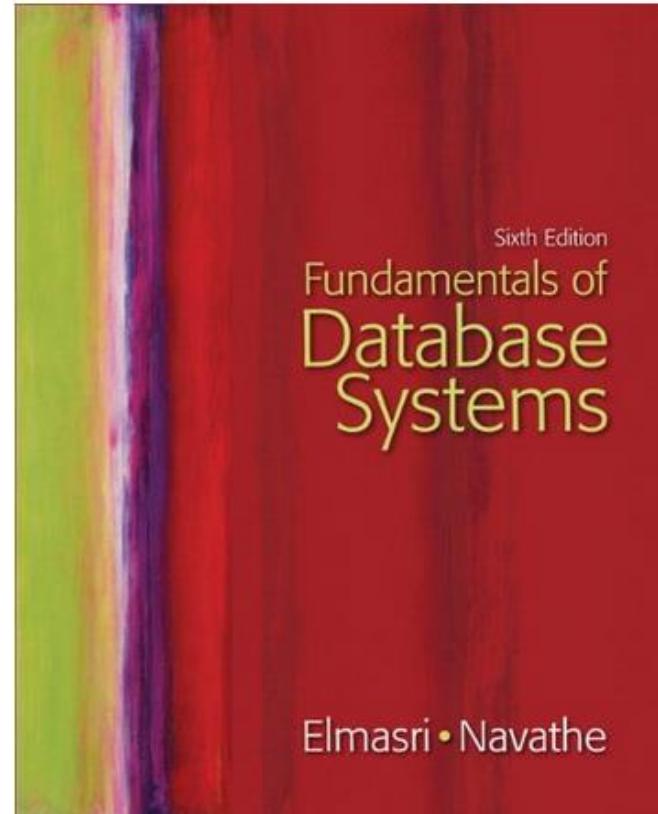
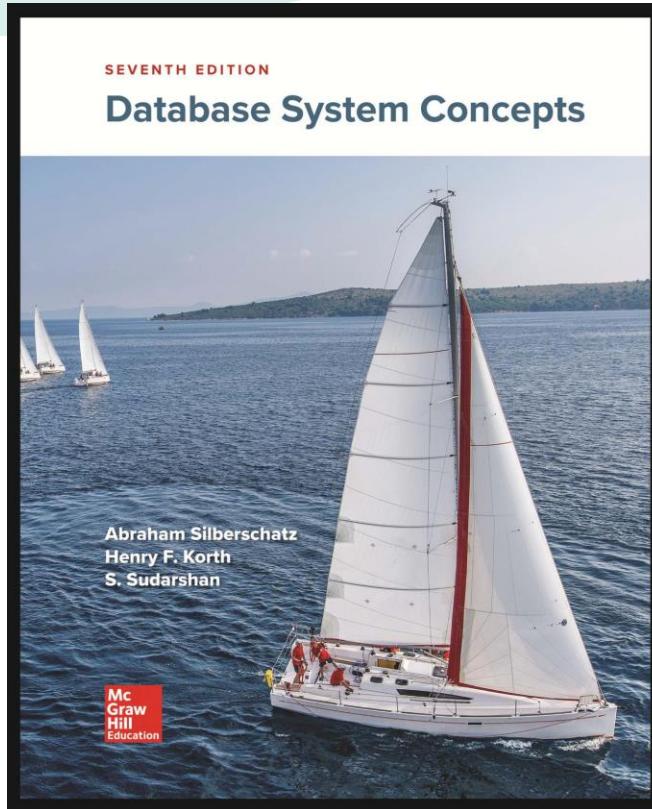
IF3140 - Sistem Basis Data

Database Security



KNOWLEDGE & SOFTWARE ENGINEERING





References

Ramez Elmasri, Shamkant B. Navathe :
"Fundamentals of Database Systems", 6th Edition

- Chapter 24 : Database Security
- Abraham Silberschatz, Henry F. Korth, S. Sudarshan : **"Database System Concepts"**, 7th Edition
- Chapter 4.7 : Authorization
- Chapter 9.8 : Application Security
- Chapter 9.9 : Encryption and Its Application



KNOWLEDGE & SOFTWARE ENGINEERING

Objectives



KNOWLEDGE & SOFTWARE ENGINEERING



Understand important properties of security in a Database System



Evaluate access controls of a specified database by using authorization-grant graph



Manage users of databases with specified access controls related with a case study

Outline

Database Security and Authorization

Access Control

Introduction to Statistical Database Security

Introduction to Flow Control

Encryption and Public Key Infrastructures

Application Security



KNOWLEDGE & SOFTWARE ENGINEERING

Database Security and Authorization



KNOWLEDGE & SOFTWARE ENGINEERING

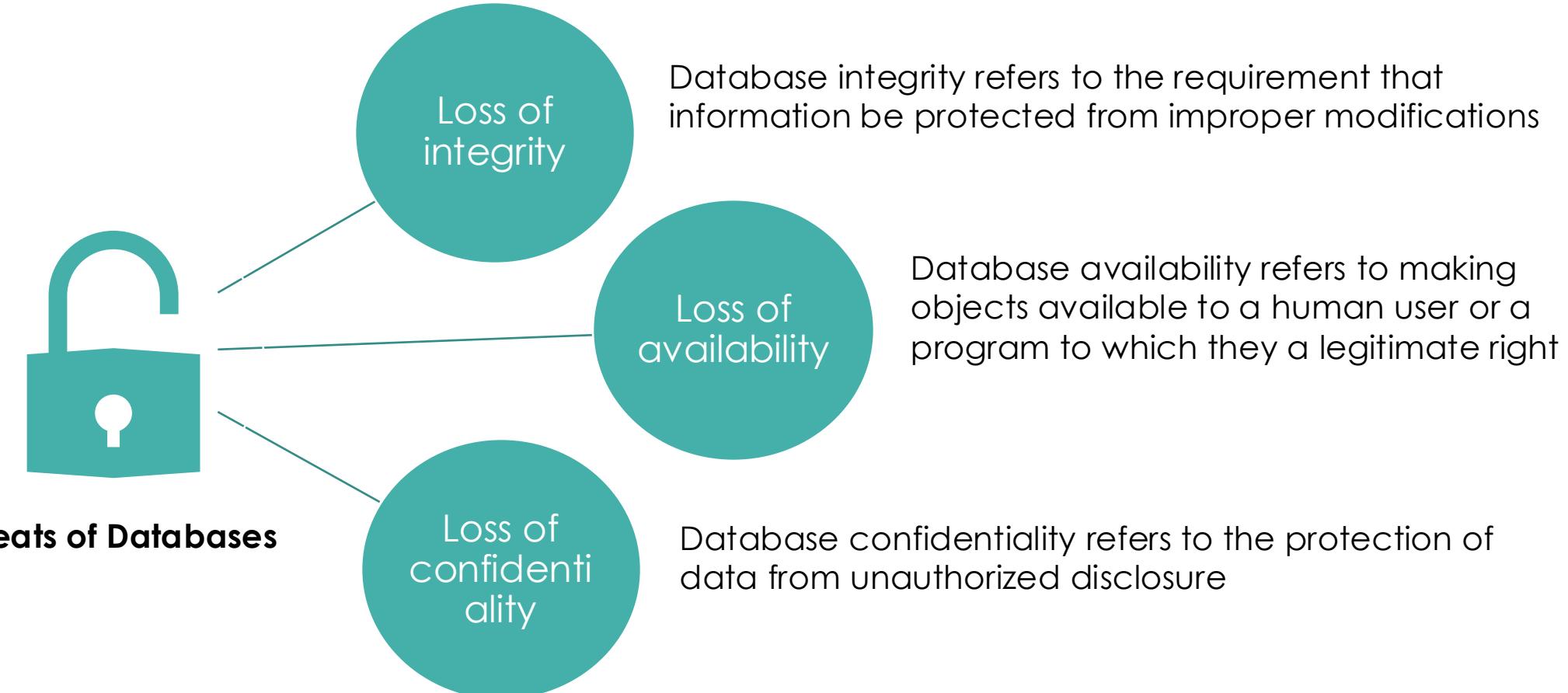
Introduction to Database Security Issues (1/4)

Database security addresses many issues, including the following

- Various **legal and ethical issues** regarding the right to access certain information
- **Policy issues** at the governmental, institutional, or corporate level to what kinds of information should not be made publicly available
- **System-related issues** such as the system levels at which various security functions should be enforced
- The need in some organizations to **identify multiple security levels** and to categorize the data and users based on these classifications

Introduction to Database Security

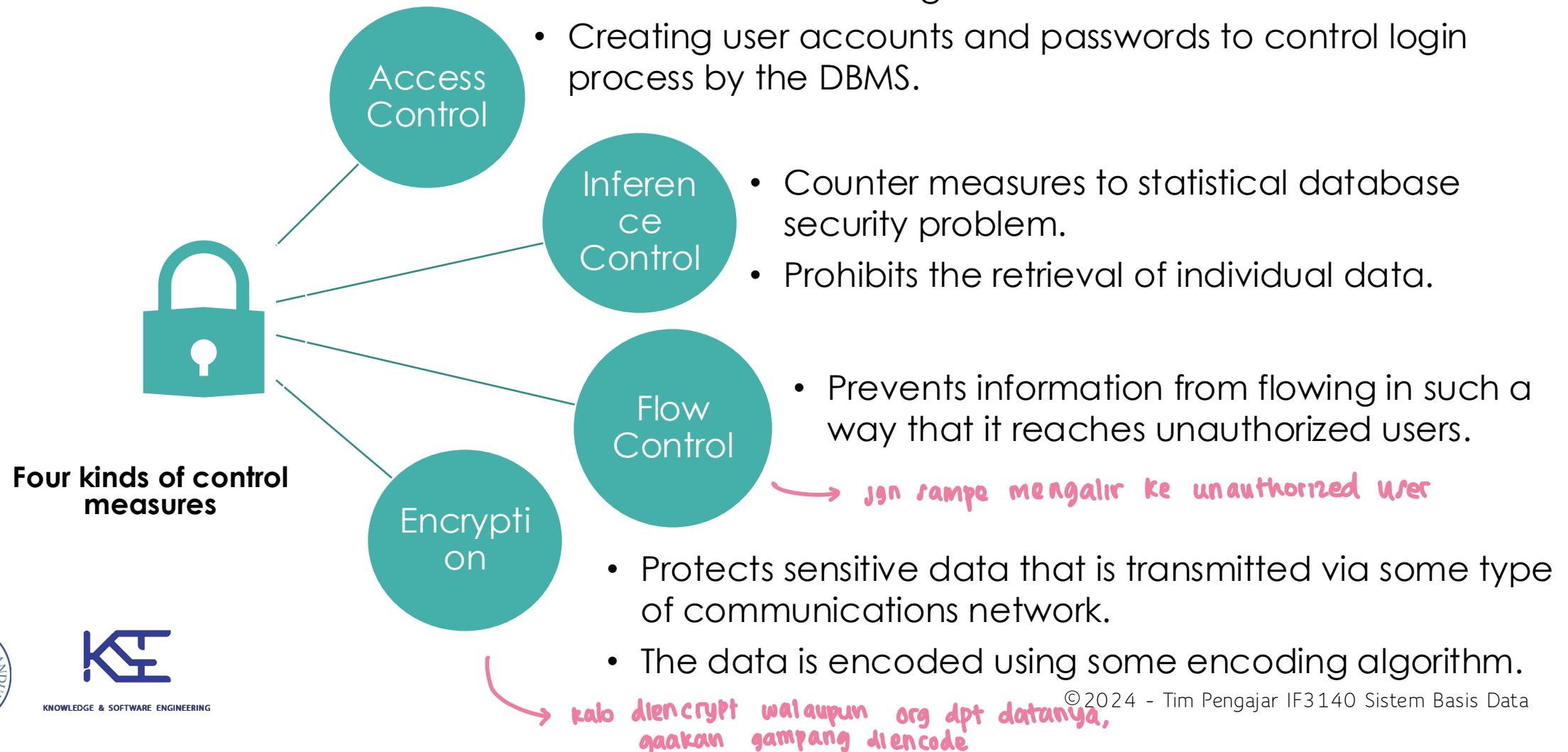
Issues (2/4)



Introduction to Database Security

Issues (3/4)

8



KNOWLEDGE & SOFTWARE ENGINEERING

Introduction to Database Security Issues (4/4)

kasih privilege
ke user

A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access.



Two types of database security mechanisms:

Discretionary security mechanisms

grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).

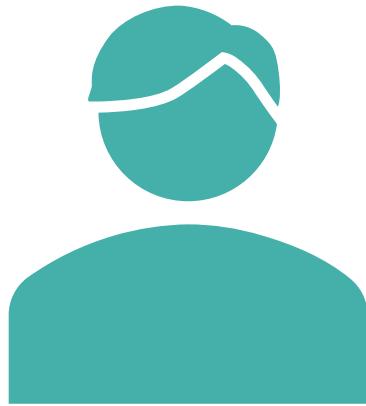
Mandatory security mechanism

enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization.



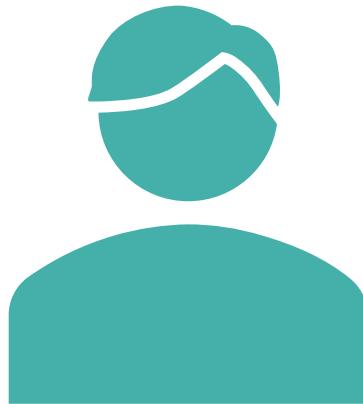
KNOWLEDGE & SOFTWARE ENGINEERING

Database Security and the DBA (1/2)



- The database administrator (DBA) is the **central authority for managing a database system**. Responsible for the overall security of the database system.
- The DBA's responsibilities include
 - **granting privileges** to users who need to use the system
 - **classifying users and data** in accordance with the policy of the organization

Database Security and the DBA (2/2)



- The DBA has a DBA account in the DBMS (Sometimes these are called a system or superuser account)
 - These accounts **provide powerful capabilities** such as:
 1. Account creation Access Control
 2. Privilege granting Discretionary
 3. Privilege revocation Discretionary
 4. Security level assignment Mandatory Security

Access Protection, User Accounts, and Database Audits

12

Whenever a person or group of persons need to access a database system, the individual or group must **first apply for a user account**.

- The DBA will then create a new account id and password for the user

The user must log in to the DBMS by **entering account id and password** whenever database access is needed.

The database system must **keep track of all operations** on the database that are applied by a certain user throughout each login session.

- To keep a record of all updates applied to the database and of the particular user who applied each update, that may be required for recovery from a transaction failure or system crash.

If any tampering with the database is suspected, a **database audit** is performed

- A **database audit** consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period.



KNOWLEDGE & SOFTWARE ENGINEERING

harus ditrack user apa
 bisa akses data apa
 pas kapan

© 2024 - Tim Pengajar IF3140 Sistem Basis Data

Access Control

Discretionary Access Control (DAC)



KNOWLEDGE & SOFTWARE ENGINEERING

Types of Discretionary Privileges (I/2)

The typical method of enforcing discretionary access control in a database system is based on the granting and revoking privileges.

atur siapa yg bisa melakukan ini

The **account level**; the capabilities provided to the account itself and can include:

- the **CREATE SCHEMA** or **CREATE TABLE** privilege, to create a schema or base relation;
- the **CREATE VIEW** privilege;
- the **ALTER** privilege, to apply schema changes such adding or removing attributes from relations;
- the **DROP** privilege, to delete relations or views;
- the **MODIFY** privilege, to insert, delete, or update tuples;
- and the **SELECT** privilege, to retrieve information from the database by using a **SELECT** query.

The **relation level** (or **table level**, includes **base relations** and virtual (**view**) relations):

- At this level, the DBA can control the privilege to access each individual relation or view in the database.

Types of Discretionary Privileges (2/2)

Granting and revoking privilege generally follow an authorization model for discretionary privileges known as access matrix model.

Columns of matrix M represents objects (relations, records, columns, views, operations)

	Relation X	Relation Y	View Z
User A	Read	Owner	Owner
User B	Owner		Read
Program C	Read, write	Read, write	

Rows of matrix M
represents subjects
(users, accounts,
programs)

$M(i,j)$ represents the type of privileges (read, write,
update) that subject i holds on object j



KNOWLEDGE & SOFTWARE ENGINEERING

Access Control

Mandatory Access Control (MAC)



KNOWLEDGE & SOFTWARE ENGINEERING

Mandatory Access Control and Role-Based Access Control for Multilevel Security

- The discretionary access control techniques of granting and revoking privileges on relations has traditionally been the main security mechanism for relational database systems.
- This is an all-or-nothing method:
 - A user either has or does not have a certain privilege.
- In many applications, an **additional security policy** is needed that classifies data and users based on security classes.
 - This approach as **mandatory access control**, would typically be **combined** with the discretionary access control mechanisms.

Mandatory Access Control and Role-Based Access Control for Multilevel Security(2)

- Typical **security classes** are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest:

$$TS \geq S \geq C \geq U$$

semua orang bisa akses

- The commonly used model for multilevel security, known as the Bell-LaPadula model, classifies each **subject** (user, account, program) and **object** (relation, tuple, column, view, operation) into one of the security classifications, TS, S, C, or U:
 - **Clearance** (classification) of a subject S as **class(S)** and to the **classification** of an object O as **class(O)**.



Mandatory Access Control and Role-Based Access Control for Multilevel Security(3)

- Two restrictions are enforced on data access based on the subject/object classifications:
 - **Simple security property:** A subject S is not allowed read access to an object O unless $\text{class}(S) \geq \text{class}(O)$.
 - A subject S is not allowed to write an object O unless $\text{class}(S) \leq \text{class}(O)$. This known as the **star property** (or * property).

contoh

$S_x = \text{subject } x$

$O_x = \text{object } x$

	Read	Write
$\text{class}(S_1) = S$		
$\text{class}(O_1) = S$	✓	✓
$\text{class}(O_2) = U$	✓	✗
$\text{class}(O_3) = TS$	✗	✓



Mandatory Access Control and Role-Based Access Control for Multilevel Security(4)

Example:

(a) EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Brown C	80000 S	Good C	S

(b) EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	NULL C	C
Brown C	NULL C	Good C	C

(c) EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	NULL U	NULL U	U

A multilevel relation to illustrate multilevel security. (a)

The original EMPLOYEE tuples. (b) Appearance of EMPLOYEE after filtering for classification C users.

(c) Appearance of EMPLOYEE after filtering for classification U users.



Comparing Discretionary Access Control and Mandatory Access Control

In many practical situations, discretionary policies are preferred because they offer a better trade-off between security and applicability.



Policies	Advantages	Drawbacks
Discretionary Access Control (DAC)	Characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains.	Main drawback of DAC models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs.
Mandatory Access Control (MAC) <i>harus dibuat sendiri aplikasinya</i>	Ensure a high degree of protection in a way, they prevent any illegal flow of information.	Mandatory policies have the drawback of being too rigid and they are only applicable in limited environments.



Access Control

Role-Based Access Control (RBAC)



KNOWLEDGE & SOFTWARE ENGINEERING

Role-Based Access Control

- **Role-based access control (RBAC)** emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprise wide systems.
- **Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles.**
- Roles can be created using the **CREATE ROLE** and **DESTROY ROLE** commands.
 - The **GRANT** and **REVOKE** commands discussed under DAC can then be used to assign and revoke privileges from roles.

```
GRANT ROLE full_time TO employee_type1
GRANT ROLE intern TO employee_type2
```



KNOWLEDGE & SOFTWARE ENGINEERING

Role-Based Access Control(2)

- **RBAC** appears to be a viable alternative to traditional discretionary and mandatory access controls; it **ensures that only authorized users are given access to certain data or resources.**
- Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.
- Role hierarchy in **RBAC** is a natural way of organizing roles to reflect the organization's lines of authority and responsibility.

Role-Based Access Control(3)

- Another important consideration in **RBAC** systems is **the possible temporal constraints that may exist on roles**, such as time and duration of role activations, and timed triggering of a role by an activation of another role.
- Using an **RBAC** model is **highly desirable goal for addressing the key security requirements of Web-based applications**.
- In contrast, discretionary access control (**DAC**) and mandatory access control (**MAC**) models **lack capabilities** needed to support the security requirements emerging enterprises and Web-based applications.

Access Control in SQL

SLIDES ARE TAKEN FROM: DATABASE SYSTEMS CONCEPTS, 7TH EDITION

©SILBERSCHATZ, KORTH, SUDARSHAN



KNOWLEDGE & SOFTWARE ENGINEERING

Authorization

Forms of authorization on parts of the database:

- **Read** - allows reading, but not modification of data.
- **Insert** - allows insertion of new data, but not modification of existing data.
- **Update** - allows modification, but not deletion of data.
- **Delete** - allows deletion of data.

Forms of authorization to modify the database schema

- **Index** - allows creation and deletion of indices.
- **Resources** - allows creation of new relations.
- **Alteration** - allows addition or deletion of attributes in a relation.
- **Drop** - allows deletion of relations.

Authorization Specification in SQL

- The **grant** statement is used to confer authorization

```
grant <privilege list>
on <relation name or view name> to <user list>
```

- <user list> is:
 - a user-id
 - **public**, which allows all valid users the privilege granted
 - A role
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- **The grantor of the privilege must already hold the privilege on the specified item** (or be the database administrator).

Privileges in SQL

- **select**: allows read access to relation, or the ability to query using the view
 - Example: grant users U_1 , U_2 , and U_3 **select** authorization on the *instructor* relation:

```
grant select on instructor to  $U_1$ ,  $U_2$ ,  $U_3$ 
```

- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges



KNOWLEDGE & SOFTWARE ENGINEERING

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

```
revoke <privilege list>
on <relation name or view name> to <user list>
```

- Example:

```
revoke select on branch from U1, U2, U3
```

- <privilege-list> may be **all** to revoke all privileges the revoke may hold.
- If <revoke-list> includes **public**, all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Roles

```
create role instructor;  
grant instructor to Amit;
```

- Privileges can be granted to roles:
`grant select on takes to instructor;`
- Roles can be granted to users, as well as to other roles

```
create role teaching_assistant  
grant teaching_assistant to instructor;  
Instructor inherits all privileges of teaching_assistant
```

- Chain of roles

```
create role dean;  
grant instructor to dean;  
grant dean to Satoshi;
```



KNOWLEDGE & SOFTWARE ENGINEERING

Authorization on Views

- `create view geo_instructor as
(select *
from instructor
where dept_name = 'Geology');`
- `grant select on geo_instructor to geo_staff`
- Suppose that a `geo_staff` member issues
 - `select *`
`from geo_instructor;`
 - What if
 - `geo_staff` does not have permissions on `instructor`?
 - creator of view did not have some permissions on `instructor`?

Other Authorization Features

- **references** privilege to create foreign key

```
grant reference (dept_name) on department to Mariano;
```

- **transfer** of privileges

```
grant select on department to Amit with grant option;
```



KNOWLEDGE & SOFTWARE ENGINEERING

Other Authorization Features

- **revoking** of privileges

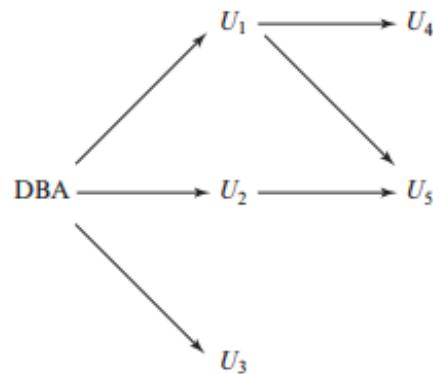


Figure 4.11 Authorization-grant graph (U_1, U_2, \dots, U_5 are users and DBA refers to the database administrator).

revoke select on department from Amit, Satoshi cascade;

revoke select on department from Amit, Satoshi restrict;

Etc. read Section 4.7 for more details we have omitted here.

Access Control Policies for E-Commerce and the Web

- **E-Commerce environments** require elaborate policies that go beyond traditional DBMSs.
 - In an e-commerce environment the resources to be protected are not only traditional data but also knowledge and experience.
 - The access control mechanism should be flexible enough to support a wide spectrum of heterogeneous protection objects.
- A related requirement is the support for **content-based access-control**.

Access Control Policies for E-Commerce and the Web(2)

- Another requirement is related to the heterogeneity of subjects, which requires access control policies based on user characteristics and qualifications.
 - A possible solution, to better take into account user profiles in the formulation of access control policies, is to support the notion of credentials.
 - A **credential** is a set of properties concerning a user that are relevant for security purposes
 - For example, age, position within an organization
 - It is believed that the XML language can play a key role in access control for e-commerce applications.

Introduction to Statistical Database Security



KNOWLEDGE & SOFTWARE ENGINEERING

Introduction to Statistical Database Security

- **Statistical databases** are used mainly to produce statistics on various populations.
- The database may contain **confidential data** on individuals, which should be protected from user access.
- Users are permitted to retrieve **statistical information** on the populations, such as **averages, sums, counts, maximums, minimums**, and **standard deviations**.
- A **population** is a set of tuples of a relation (table) that satisfy some selection condition.
- Statistical queries involve applying **statistical functions** to a **population** of tuples.

Introduction to Statistical Database Security(2)

- For example, we may want to retrieve the number of individuals in a **population** or the average *income* in the population.
 - However, statistical users are not allowed to retrieve individual data, such as the income of a specific person.
- Statistical database security techniques must prohibit the retrieval of individual data.
- This can be achieved by prohibiting queries that retrieve attribute values and by allowing only queries that involve statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION.
 - Such queries are sometimes called **statistical queries**.

Introduction to Statistical Database Security(3)

- It is DBMS's responsibility to ensure confidentiality of information about individuals, while still providing useful statistical summaries of data about those individuals to users. Provision of **privacy protection** of users in a statistical database is paramount.
- In some cases it is possible to **infer** the values of individual tuples from a sequence of statistical queries.
 - This is particularly true when the conditions result in a population consisting of a small number of tuples.



KNOWLEDGE & SOFTWARE ENGINEERING

Introduction to Flow Control



KNOWLEDGE & SOFTWARE ENGINEERING

Introduction to Flow Control

- **Flow control** regulates the distribution or flow of information among accessible objects.
- A **flow** between object X and object Y occurs when a program reads values from X and writes values into Y.
 - Flow controls check that information contained in some objects does not flow explicitly or implicitly into less protected objects.
- A **flow policy** specifies the channels along which information is allowed to move.
 - The simplest flow policy specifies just two classes of information:
 - confidential (C) and nonconfidential (N)
 - and allows all flows except those from class C to class N.

Covert Channels

- A **covert channel** allows a transfer of information that violates the security or the policy.
- A **covert channel allows** information to pass from a higher classification level to a lower classification level through **improper means**.
- **Covert channels** can be classified into two broad categories:
 - **Storage channels** do not require any temporal synchronization, in that information is conveyed by accessing system information or what is otherwise inaccessible to the user.
 - **Timing channel** allow the information to be conveyed by the timing of events or processes.
- Some security experts believe that one way to avoid covert channels is for programmers to not actually gain access to sensitive data that a program is supposed to process after the program has been put into operation.



KNOWLEDGE & SOFTWARE ENGINEERING

Encryption

SLIDES ARE TAKEN FROM: DATABASE
SYSTEMS CONCEPTS, 7TH EDITION

©SILBERSCHATZ, KORTH, SUDARSHAN



KNOWLEDGE & SOFTWARE ENGINEERING

Encryption

- Data may be encrypted when database authorization provisions do not offer sufficient protection.
- Properties of good encryption technique:
 - Relatively simple for authorized users to encrypt and decrypt data.
 - Encryption scheme depends not on the secrecy of the algorithm but on the secrecy of a parameter of the algorithm called the encryption key.
 - Extremely difficult for an intruder to determine the encryption key.
- **Symmetric-key encryption**: same key used for encryption and for decryption
- **Public-key encryption** (a.k.a. **asymmentric-key encryption**): use different keys for encryption and decryption
 - encryption key can be public, decryption key secret

Encryption (Cont.)

- **Data Encryption Standard (DES)** substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism. Scheme is no more secure than the key transmission mechanism since the key has to be shared.
- **Advanced Encryption Standard (AES)** is a new standard replacing DES, and is based on the Rijndael algorithm, but is also dependent on shared secret keys.
- **Public-key encryption** is based on each user having two keys:
 - public key – publicly published key used to encrypt data, but cannot be used to decrypt data
 - private key -- key known only to individual user and used to decrypt data. Need not be transmitted to the site doing encryption.

Encryption scheme is such that it is impossible or extremely hard to decrypt data given only the public key.

- The RSA public-key encryption scheme is based on the hardness of factoring a very large number (100's of digits) into its prime components

Encryption (Cont.)

- **Hybrid schemes** combining public key and private key encryption for efficient encryption of large amounts of data
- Encryption of small values such as identifiers or names vulnerable to **dictionary attacks**
 - especially if encryption key is publicly available
 - but even otherwise, statistical information such as frequency of occurrence can be used to reveal content of encrypted data
 - Can be deterred by adding extra random bits to the end of the value, before encryption, and removing them after decryption
 - same value will have different encrypted forms each time it is encrypted, preventing both above attacks
 - extra bits are called **salt bits**

Encryption in Databases

- Database widely support encryption
- Different levels of encryption:
 - **disk block**
 - every disk block encrypted using key available in database-system software.
 - Even if attacker gets access to database data, decryption cannot be done without access to the key.
 - **Entire relations, or specific attributes of relations**
 - non-sensitive relations, or non-sensitive attributes of relations need not be encrypted
 - however, attributes involved in primary/foreign key constraints cannot be encrypted.
- Storage of encryption or decryption keys
 - typically, single master key used to protect multiple encryption/decryption keys stored in database
- Alternative: encryption/decryption is done in application, before sending values to the database

Encryption and Authentication

- Password based authentication is widely used, but is susceptible to sniffing on a network.
- **Challenge-response** systems avoid transmission of passwords
 - DB sends a (randomly generated) challenge string to user.
 - User encrypts string and returns result.
 - DB verifies identity by decrypting result
 - Can use public-key encryption system by DB sending a message encrypted using user's public key, and user decrypting and sending the message back.
- **Digital signatures** are used to verify authenticity of data
 - E.g., use private key (in reverse) to encrypt data, and anyone can verify authenticity by using public key (in reverse) to decrypt data. Only holder of private key could have created the encrypted data.
 - Digital signatures also help ensure **nonrepudiation**: sender cannot later claim to have not created the data

Application Security

SLIDES ARE TAKEN FROM: DATABASE
SYSTEMS CONCEPTS, 7TH EDITION

©SILBERSCHATZ, KORTH, SUDARSHAN



KNOWLEDGE & SOFTWARE ENGINEERING

SQL Injection

- Suppose query is constructed using
 - "select * from instructor where name = "" + name + """
- Suppose the user, instead of entering a name, enters:
 - X' or 'Y' = 'Y
- then the resulting statement becomes:
 - "select * from instructor where name = "" + "X' or 'Y' = 'Y" + """
 - which is:
 - select * from instructor where name = 'X' or 'Y' = 'Y'
- User could have even used
 - X'; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:
"select * from instructor where name = 'X\' or \'Y\' = \'Y'"
- **Always use prepared statements, with user inputs as parameters**
- Is the following prepared statement secure?
 - conn.prepareStatement("select * from instructor where name = "" + name + "")")



KNOWLEDGE & SOFTWARE ENGINEERING

Cross Site Scripting

- HTML code on one page executes action on another page
 - E.g. <img src = <http://mybank.com/transfermoney?amount=1000&toaccount=14523>>
 - Risk: if user viewing page with above code is currently logged into mybank, the transfer may succeed
 - Above example simplistic, since GET method is normally not used for updates, but if the code were instead a script, it could execute POST methods
- Above vulnerability called **cross-site scripting (XSS)** or **cross-site request forgery (XSRF or CSRF)**
- **Prevent your web site from being used to launch XSS or XSRF attacks**
 - Disallow HTML tags in text input provided by users, using functions to detect and strip such tags
- **Protect your web site from XSS/XSRF attacks launched from other sites**
 - ..next slide

Cross Site Scripting

- **Protect your web site from XSS/XSRF attacks launched from other sites**
 - Use **referer** value (URL of page from where a link was clicked) provided by the HTTP protocol, to check that the link was followed from a valid page served from same site, not another site
 - Ensure IP of request is same as IP from where the user was authenticated
 - prevents hijacking of cookie by malicious user
 - Never use a GET method to perform any updates
 - This is actually recommended by HTTP standard

Password Leakage

- Never store passwords, such as database passwords, in clear text in scripts that may be accessible to users
 - E.g. in files in a directory accessible to a web server
 - Normally, web server will execute, but not provide source of script files such as file.jsp or file.php, but source of editor backup files such as file.jsp~, or .file.jsp.swp may be served
- Restrict access to database server from IPs of machines running application servers
 - Most databases allow restriction of access by source IP address

Application Authentication

- Single factor authentication such as passwords too risky for critical applications
 - guessing of passwords, sniffing of packets if passwords are not encrypted
 - passwords reused by user across sites
 - spyware which captures password
- Two-factor authentication
 - e.g. password plus one-time password sent by SMS
 - e.g. password plus one-time password devices
 - device generates a new pseudo-random number every minute, and displays to user
 - user enters the current number as password
 - application server generates same sequence of pseudo-random numbers to check that the number is correct.

Application Authentication

- **Man-in-the-middle** attack
 - E.g. web site that pretends to be mybank.com, and passes on requests from user to mybank.com, and passes results back to user
 - Even two-factor authentication cannot prevent such attacks
- Solution: authenticate Web site to user, using digital certificates, along with secure http protocol
- **Central authentication** within an organization
 - application redirects to central authentication service for authentication
 - avoids multiplicity of sites having access to user's password
 - LDAP or Active Directory used for authentication

Single Sign-On

- **Single sign-on** allows user to be authenticated once, and applications can communicate with authentication service to verify user's identity without repeatedly entering passwords
- **Security Assertion Markup Language (SAML)** standard for exchanging authentication and authorization information across security domains
 - e.g. user from Yale signs on to external application such as acm.org using userid joe@yale.edu
 - application communicates with Web-based authentication service at Yale to authenticate user, and find what the user is authorized to do by Yale (e.g. access certain journals)
- **OpenID** standard allows sharing of authentication across organizations
 - e.g. application allows user to choose Yahoo! as OpenID authentication provider, and redirects user to Yahoo! for authentication

Application-Level Authorization

- Current SQL standard does not allow fine-grained authorization such as “students can see their own grades, but not other’s grades”
 - Problem 1: Database has no idea who are application users
 - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

```
create view studentTakes as
select *
from takes
where takes.ID = syscontext.user_id()
```

 - where syscontext.user_id() provides end user identity
 - end user identity must be provided to the database by the application
 - Having multiple such views is cumbersome

Application-Level Authorization (Cont.)

- Currently, authorization is done entirely in application
- Entire application code has access to entire database
 - large surface area, making protection harder
- Alternative: **fine-grained (row-level) authorization** schemes
 - extensions to SQL authorization proposed but not currently implemented
 - Oracle Virtual Private Database (VPD) allows predicates to be added transparently to all SQL queries, to enforce fine-grained authorization
 - e.g. add `ID= sys_context.user_id()` to all queries on student relation if user is a student



KNOWLEDGE & SOFTWARE ENGINEERING

Audit Trails

- Applications must log actions to an audit trail, to detect who carried out an update, or accessed some sensitive data
- Audit trails used after-the-fact to
 - detect security breaches
 - repair damage caused by security breach
 - trace who carried out the breach
- Audit trails needed at
 - Database level, and at
 - Application level

End of Chapter

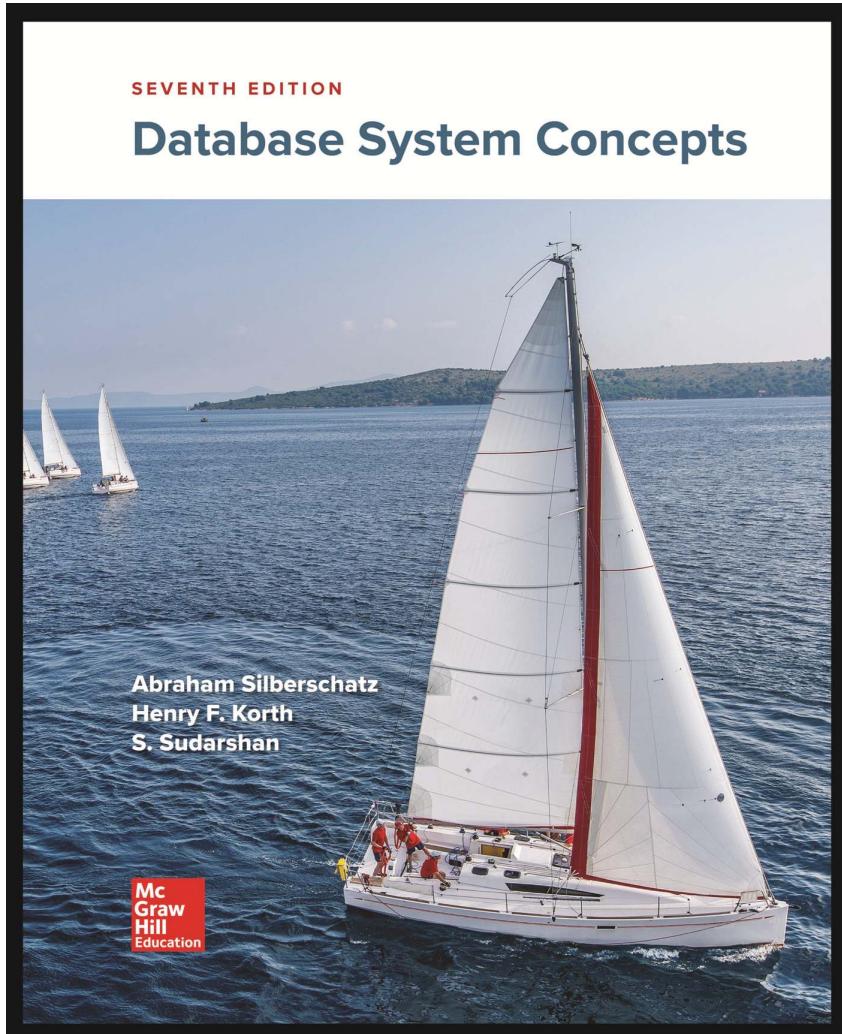


KNOWLEDGE & SOFTWARE ENGINEERING

IF3140 – Sistem Basis Data Transactions

SEMESTER I TAHUN AJARAN 2024/2025





KNOWLEDGE & SOFTWARE ENGINEERING

© Tim Pengajar IF3140 Semester 1 2024/2025

Sumber

Silberschatz, Korth, Sudarshan:
“Database System Concepts”,
7th Edition

- Chapter 17: Transactions

Objectives



Students are able to:

- Explain the importance of transaction properties
- Explain serializable transactions
- Explain the concept of implicit commits
- Describe the issues specific to efficient transaction execution
- Explain at least two transaction protocols

Outline



KNOWLEDGE & SOFTWARE ENGINEERING



- Transaction Concept
- Transaction State
- Concurrent Executions
- Serializability
- Recoverability
- Implementation of Isolation
- Transaction Definition in SQL
- Testing for Serializability

Transaction Concept

A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.

Example

```
1.read(A)  
2.A := A - 50  
3.write(A)  
4.read(B)  
5.B := B + 50  
6.write(B)
```

Main issues

- Failures of various kinds
- Concurrent execution of multiple transactions



KNOWLEDGE & SOFTWARE ENGINEERING

Example of Fund Transfer

Transaction to transfer \$50 from account A to account B:

1. **read(A)**
2. $A := A - 50$
3. **write(A)**
4. **read(B)**
5. $B := B + 50$
6. **write(B)**

- **Atomicity requirement**

- if the transaction fails after step 3 and before step 6, money will be “lost” leading to an inconsistent database state
 - Failure could be due to software or hardware
- the system should ensure that updates of a partially executed transaction are not reflected in the database

- **Durability requirement** — once the user has been notified that the transaction has completed (i.e., the transfer of the \$50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures.



KNOWLEDGE & SOFTWARE ENGINEERING

Example of Fund Transfer (Cont.)

Transaction to transfer \$50 from account A to account B:

1. **read(A)**
2. $A := A - 50$
3. **write(A)**
4. **read(B)**
5. $B := B + 50$
6. **write(B)**

• **Consistency requirement** in example:

- The sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include:
 - Explicitly specified integrity constraints such as primary keys and foreign keys
 - Implicit integrity constraints
 - e.g. sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
- A transaction must see a consistent database.
- During transaction execution the database may be temporarily inconsistent.
- When the transaction completes successfully the database must be consistent
- Erroneous transaction logic can lead to inconsistency



KNOWLEDGE & SOFTWARE ENGINEERING

Example of Fund Transfer (Cont.)

T_1

1. read(A)
2. $A := A - 50$
3. write(A)
4. read(B)
5. $B := B + 50$
6. write(B)

T_2

- read(A), read(B), print(A+B)

- Isolation can be ensured trivially by running transactions **serially**
 - that is, one after the other.
- However, executing multiple transactions concurrently has significant benefits, as we will see later.

Isolation requirement — if between steps 3 and 6, another transaction T_2 is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).



KNOWLEDGE & SOFTWARE ENGINEERING

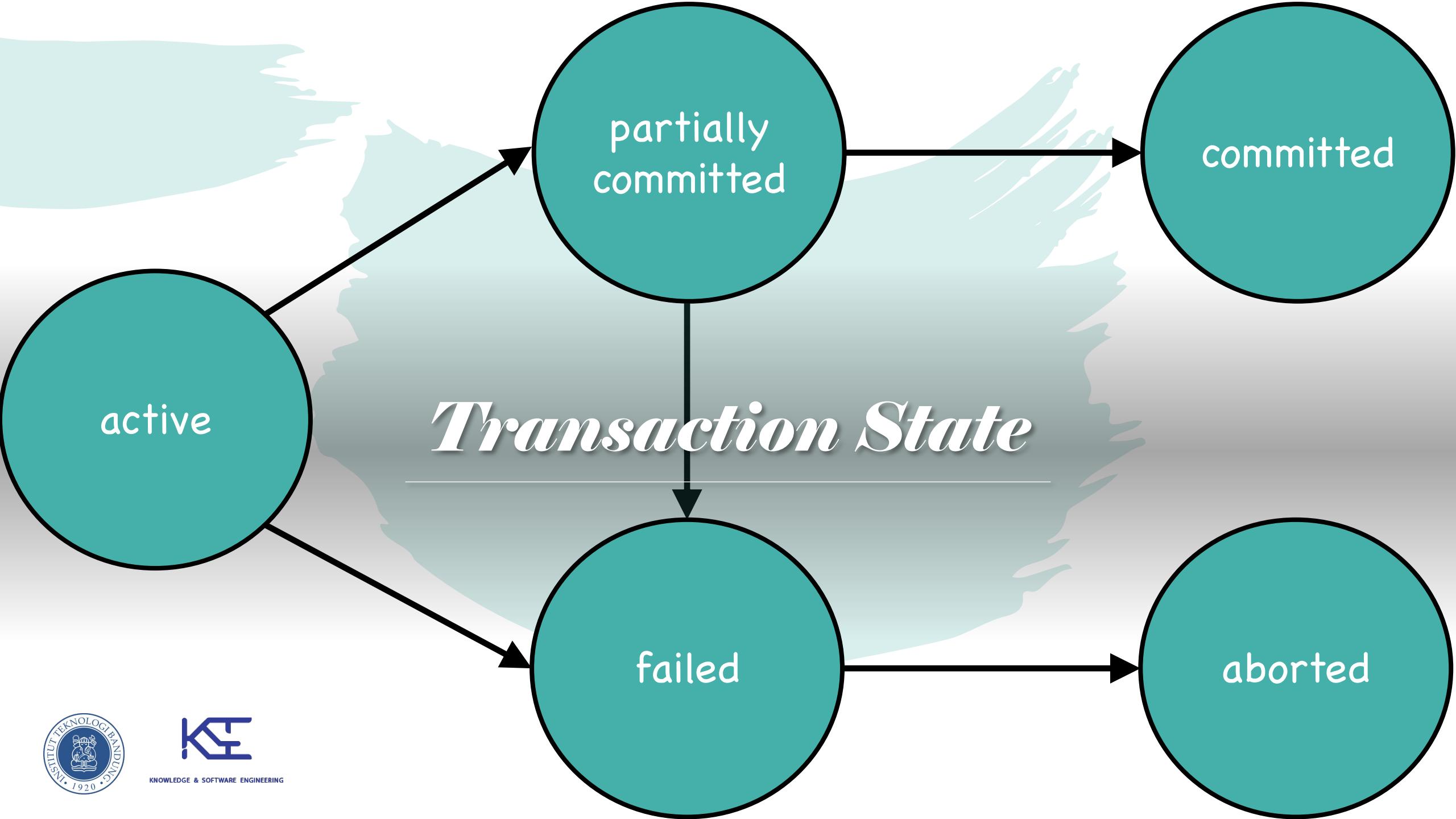
ACID Properties

A **transaction** is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

- **Atomicity**. Either all operations of the transaction are properly reflected in the database or none are.
- **Consistency**. Execution of a transaction in isolation preserves the consistency of the database.
- **Isolation**. Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
 - That is, for every pair of transactions T_i and T_j , it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished.
- **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.



KNOWLEDGE & SOFTWARE ENGINEERING



Concurrent Executions

- Advantages are
 - **increased processor and disk utilization**, leading to better transaction throughput
 - **reduced average response time for transactions**
- **Concurrency control schemes** – mechanisms to achieve isolation



KNOWLEDGE & SOFTWARE ENGINEERING

Schedules

Schedule – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed

All instructions

Preserve the order of instructions in each transaction

A transaction that successfully completes its execution will have a commit instructions as the last statement

A transaction that fails to successfully complete its execution will have an abort instruction as the last statement



Schedule (Serial)

Let T_1 transfer \$50 from A to B, and
 T_2 transfer 10% of the balance from A to B.



KNOWLEDGE & SOFTWARE ENGINEERING

Schedule 1

T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

Schedule (Serial)

Let T_1 transfer \$50 from A to B, and T_2 transfer 10% of the balance from A to B.



KNOWLEDGE & SOFTWARE ENGINEERING

Schedule 1

T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

Schedule 2

T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

Schedule

The following schedule is not a serial schedule, but it is equivalent to Schedule 1.

Let T_1 transfer \$50 from A to B, and T_2 transfer 10% of the balance from A to B.



KNOWLEDGE & SOFTWARE ENGINEERING

Schedule 1

T_1	T_2
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit

Schedule 3

T_1	T_2
read (A) $A := A - 50$ write (A)	read (A) $temp := A * 0.1$ $A := A - temp$ write (A)
read (B) $B := B + 50$ write (B) commit	read (B) $B := B + temp$ write (B) commit

Schedule

The following concurrent schedule does not preserve the value of $(A + B)$.

Let T_1 transfer \$50 from A to B , and T_2 transfer 10% of the balance from A to B .



KNOWLEDGE & SOFTWARE ENGINEERING

Schedule 4

T_1	T_2
read (A) $A := A - 50$	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B)
write (A) read (B) $B := B + 50$ write (B) commit	$B := B + temp$ write (B) commit

Serializability

- Basic Assumption – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is **serializable** if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
 - **conflict serializability**
 - **view serializability**

Simplified view of transactions

- We ignore operations other than **read** and **write** instructions
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
- Our simplified schedules consist of only **read** and **write** instructions.



KNOWLEDGE & SOFTWARE ENGINEERING

Conflicting Instructions

- Instructions I_i and I_j of transactions T_i and T_j , respectively, **conflict** if and only if there exists some item Q accessed by both I_i and I_j , and at least one of these instructions wrote Q .
- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them.

T1	T2
read (Q)	read (Q)
read (Q)	write (Q)
write (Q)	read (Q)
write (Q)	write (Q)

T1	T2
read (Q)	read (Q)
read (Q)	write (Q)
write (Q)	read (Q)
write (Q)	write (Q)

Conflict Serializability



KNOWLEDGE & SOFTWARE ENGINEERING

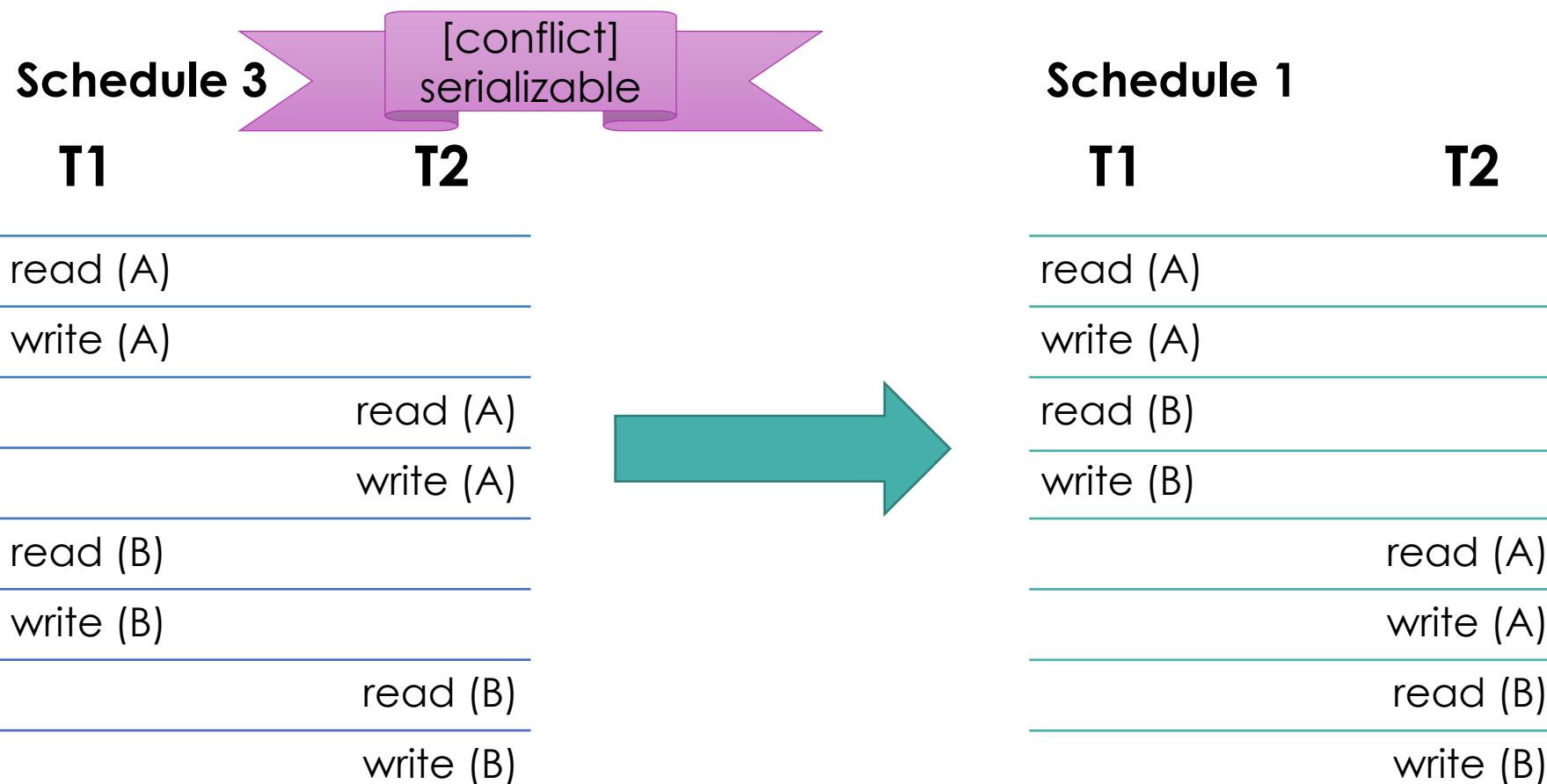
conflict equivalent schedules

schedule S can be transformed into schedule S' by a series of swaps of non-conflicting instructions

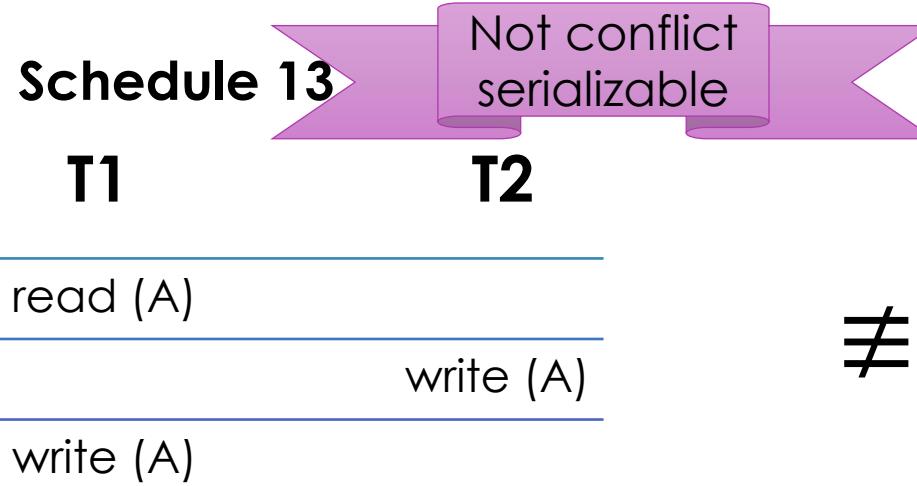
conflict serializable schedule

a schedule S that is conflict equivalent to a serial schedule

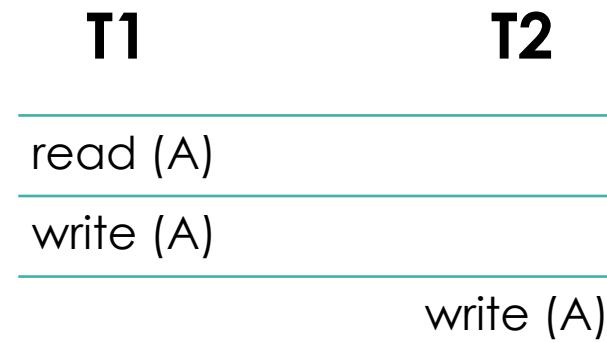
Conflict Serializability – Example 1



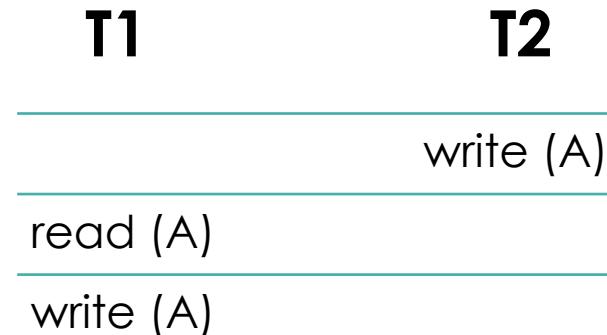
Conflict Serializability – Example 2



Schedule 11



Schedule 12



View Serializability

Let S and S' be two schedules with the same set of transactions.

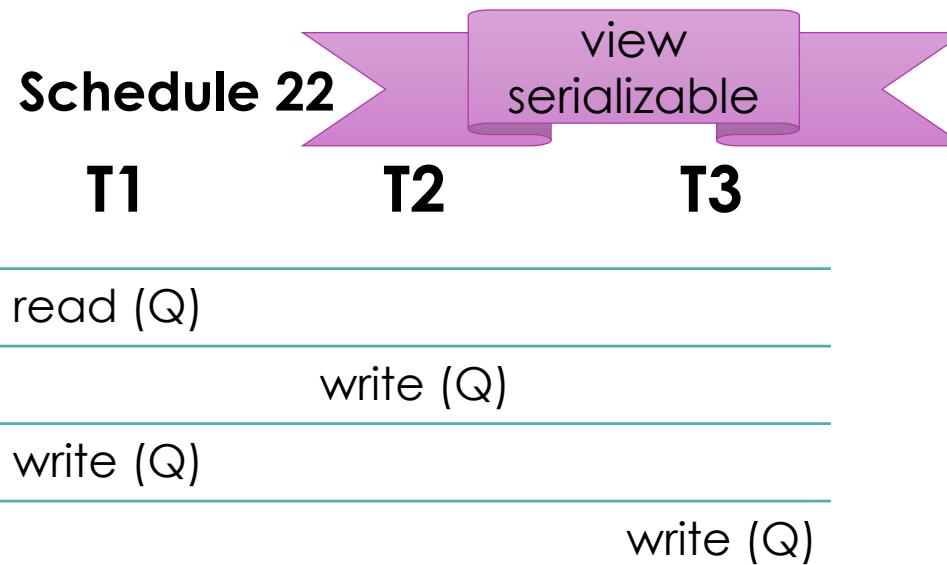
S and S' are **view equivalent** if the following three conditions are met, for each data item Q ,

- If in schedule S transaction T_i reads the initial value of Q , then in schedule S' transaction T_i must also read the initial value of Q .
- If in schedule S transaction T_i reads the value of Q that was produced by transaction T_j , then in schedule S' transaction T_i must also read the value of Q that was produced by the same **write**(Q) operation of transaction T_j .
- The transaction (if any) that performs the final write(Q) operation in schedule S must also perform the final **write**(Q) operation in schedule S' .

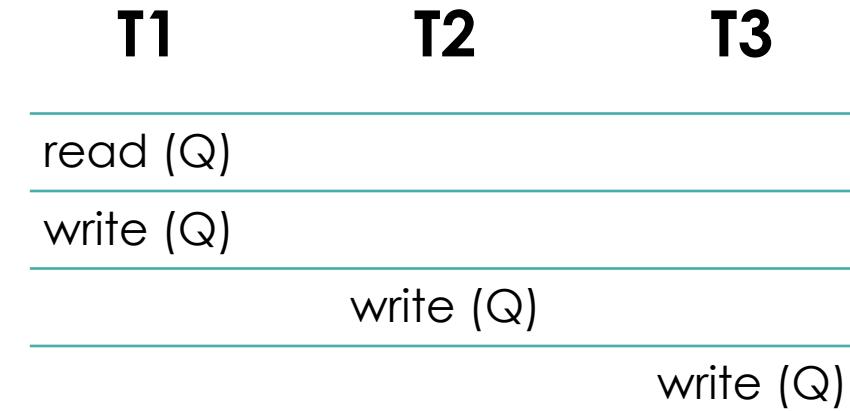
View serializable schedule: a schedule S that is view equivalent to a serial schedule.



View Serializability – Example



Schedule 21



T_1	T_5
read (A) $A := A - 50$ write (A)	
	read (B) $B := B - 10$ write (B)
read (B) $B := B + 50$ write (B)	read (A) $A := A + 10$ write (A)

Other Notions of Serializability

Determining such equivalence requires analysis of operations other than read and write



Testing for Serializability

Precedence Graph

- A directed graph
- The vertices are the transactions (names).
- An arc from T_i to T_j = the two transactions conflict, and T_i accessed earlier.
- May label the arc by the item that was accessed.

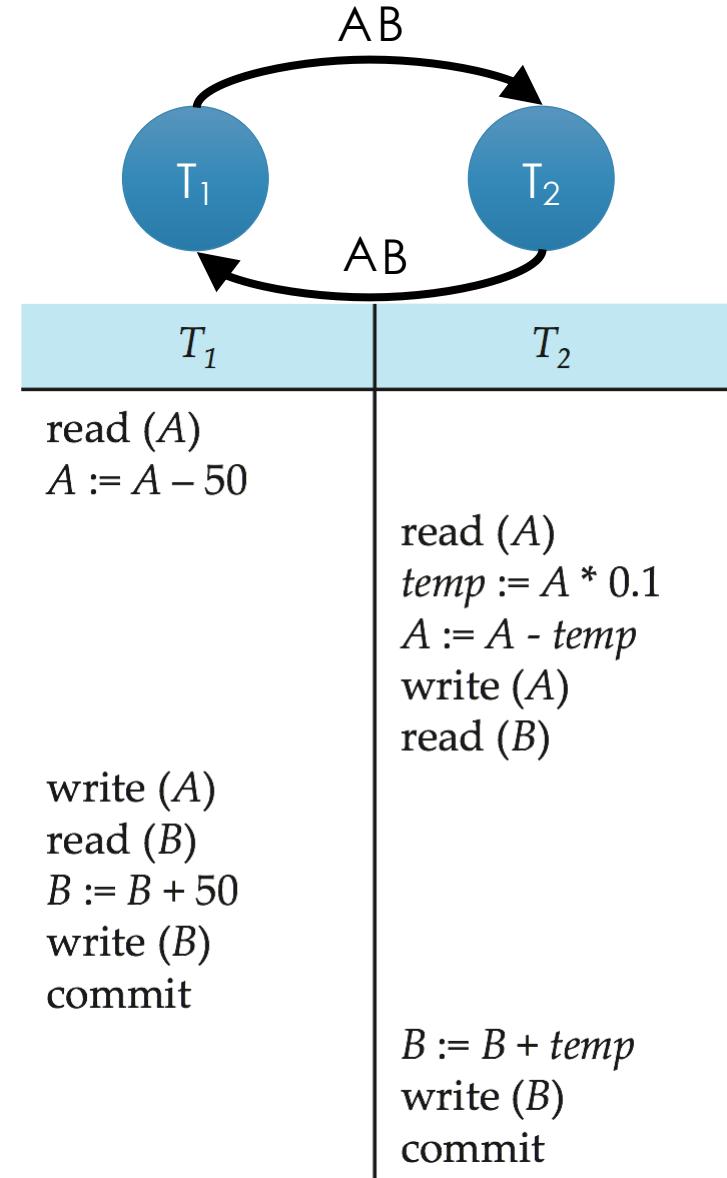


KNOWLEDGE & SOFTWARE ENGINEERING

Testing for Serializability

Precedence Graph

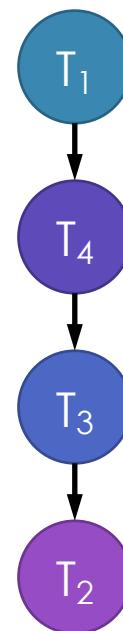
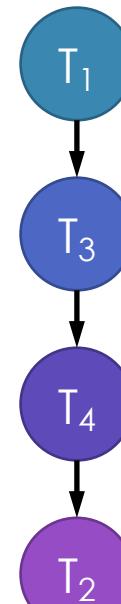
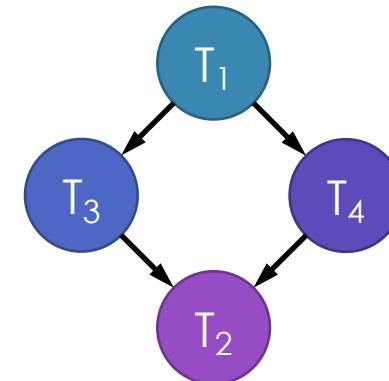
- A directed graph
- The vertices are the transactions (names).
- An arc from T_i to T_j = the two transactions conflict, and T_i accessed earlier.
- May label the arc by the item that was accessed.



Test for Conflict Serializability

A schedule is conflict serializable if and only if its **precedence graph is acyclic**.

If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph.





Test for View Serializability?

- The precedence graph test for conflict serializability cannot be used directly to test for view serializability.
 - Extension to test for view serializability has cost exponential in the size of the precedence graph.



KNOWLEDGE & SOFTWARE ENGINEERING

Recoverable Schedules

If a transaction T_j reads a data item previously written by a transaction T_i , then the commit operation of T_i appears before the commit operation of T_j .

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	
...	



Cascadeless Schedules

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

- **Cascading rollback**

a single transaction failure leads to a series of transaction rollbacks.

- **Cascadeless Schedules:** no possibility of cascading rollback

- For each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i , the commit operation of T_i appears before the read operation of T_j .
- Every cascadeless schedule is also recoverable



Concurrency Control

- A database must provide a mechanism that will ensure
 - either conflict or view serializable
 - recoverable and preferably cascadeless
- Only one transaction can be executed at a time → serial schedules
→ a poor degree of concurrency
- Testing a schedule for serializability after it has executed is a little too late!
- **Goal** – to develop concurrency control protocols that will assure serializability.



KNOWLEDGE & SOFTWARE ENGINEERING

Weak Levels of Consistency

TRADEOFF ACCURACY
FOR PERFORMANCE



KNOWLEDGE & SOFTWARE ENGINEERING

Levels of Consistency in SQL-92

Serializable — default

Repeatable read — repeated reads must return same value.

Read committed — only committed records can be read.

Read uncommitted — even uncommitted records may be read.

Read Phenomena

- The ANSI/ISO standard SQL 92 refers to three different *read phenomena* when Transaction 1 reads data that Transaction 2 might have changed:
 - Dirty Reads
 - Non-repeatable reads
 - Phantom Reads
- Suppose we have the following data:

users		
id	name	age
1	Joe	20
2	Jill	25



Dirty Reads

- aka. Uncommitted dependency:
transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed.
- Level of consistency **read uncommitted** allows dirty reads

Transaction 1

```
/* Query 1 */
SELECT age FROM users WHERE id = 1;
/* will read 20 */
```

Transaction 2

```
/* Query 2 */
UPDATE users SET age = 21 WHERE id = 1;
/* No commit here */
```

```
/* Query 1 */
SELECT age FROM users WHERE id = 1;
/* will read 21 */
```

```
ROLLBACK; /* Lock-based DIRTY READ */
```

Non-Repeatable Reads

- During the course of a transaction, a row is retrieved twice and the values within the row differ between reads
- Level of consistency **read committed** allows non-repeatable reads

Transaction 1

```
/* Query 1 */
SELECT * FROM users WHERE id = 1;
```

Transaction 2

```
/* Query 2 */
UPDATE users SET age = 21 WHERE id = 1;
COMMIT; /* in multiversion concurrency control, or lock-based READ COMMITTED */
```

```
/* Query 1 */
SELECT * FROM users WHERE id = 1;
COMMIT; /* Lock-based REPEATABLE READ */
```



KNOWLEDGE & SOFTWARE ENGINEERING

Phantom Reads

- In the course of a transaction, new rows are added by another transaction to the records being read
- Level of consistency **repeatable-read** allows phantom reads

Transaction 1

```
/* Query 1 */
SELECT * FROM users
WHERE age BETWEEN 10 AND 30;
```

Transaction 2

```
/* Query 2 */
INSERT INTO users(id,name,age) VALUES ( 3, 'Bob', 27 );
COMMIT;
```

```
/* Query 1 */
SELECT * FROM users
WHERE age BETWEEN 10 AND 30;
COMMIT;
```



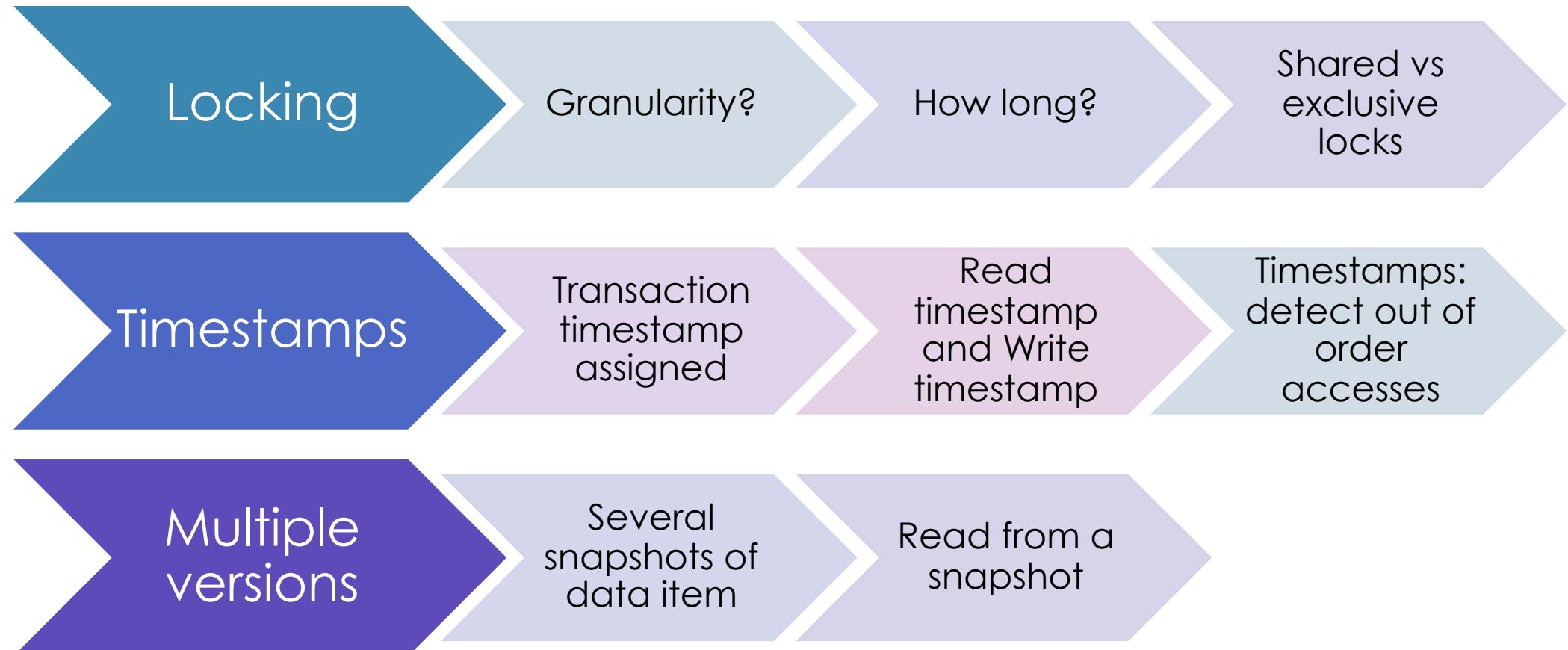
Transaction Definition in SQL

- A transaction begins implicitly
- A transaction ends by:
 - **Commit work**
 - **Rollback work**
- Every SQL statement commits implicitly
- Isolation level can be set at database level
- Isolation level can be changed at start of transaction



KNOWLEDGE & SOFTWARE ENGINEERING

Implementation of Isolation Levels



Transactions as SQL Statements

Transaction 1

```
select ID, name from instructor  
where salary > 90000
```

Transaction 2

```
insert into instructor  
values ('11111', 'James', 'Marketing', 100000)
```

Transaction 3
(with Wu's salary
= 90000)

```
update instructor set salary = salary * 1.1  
where name = 'Wu'
```

- Key idea: Detect “**predicate**” conflicts → “**predicate locking**”



End of Chapter



KNOWLEDGE & SOFTWARE ENGINEERING