



# Database System Concepts, 7<sup>th</sup> Ed.

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use

## Chapter 1: Introduction

Morteza Zakeri, Ph.D.  
(m-zakeri@live.com)

Fall 2023



# Outline

- Database-systems
- Database-systems Applications
- Purpose of Database Systems
- Our Database Course
- View of Data
- Database Languages
- Database Design
- Database Engine (RDMS)
- Transaction Management (ACID)
- Database Architecture
- Database Users and Administrators
- History of Database Systems
- OLTP and OLAP



# Database Systems

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use
- Database systems are used to manage collections of data that are:
  - Highly valuable
  - Relatively large
  - Accessed by multiple users and applications, often at the same time.
- A modern database system is a complex software system whose task is to manage a large, complex collection of data.
- Databases touch all aspects of our lives



# Database Applications Examples

- Enterprise Information
  - Sales: customers, products, purchases
  - Accounting: payments, receipts, assets
  - Human Resources: Information about employees, salaries, payroll taxes.
- Manufacturing: management of production, inventory, orders, supply chain.
- Banking and finance
  - customer information, accounts, loans, and banking transactions.
  - Credit card transactions
  - Finance: sales and purchases of financial instruments (e.g., stocks and bonds; storing real-time market data)
- Universities: registration, grades



# Database Applications Examples (Cont.)

- Airlines: reservations, schedules
- Telecommunication: records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards
- Web-based services
  - Online retailers: order tracking, customized recommendations
  - Online advertisements
- Document databases
- Navigation systems: For maintaining the locations of various places of interest along with the exact routes of roads, train systems, buses, etc.



# Purpose of Database Systems

In the early days, database applications were built directly on top of file systems, which leads to:

- Data redundancy and inconsistency: data is stored in multiple file formats resulting in duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation
  - Multiple files and formats
- Integrity problems
  - Integrity constraints (e.g.,  $\text{account balance} > 0$ ) become “buried” in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones



# Purpose of Database Systems (Cont.)

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Ex: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- Security problems
  - Hard to provide user access to some, but not all, data

**Database systems offer solutions to all the above problems**



# University Database Example

- In this course, we will be using a **university database** to illustrate all the concepts
- Data consists of information about:
  - Students
  - Instructors
  - Classes
- Application program examples:
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts





# Our Database Course

1. Relational Data Models (1 week)
2. Relational Algebra (1 week)
3. Structured Query Language (SQL) (3 weeks)
4. Database Design (2 week)
5. Database Normalization (2 week)
6. Complex Data Types (1 week)
7. Application Development (2 week)
8. Advanced Topics (Big data and NoSQL DBs) (2 week)
9. Exams (1 week)



# References

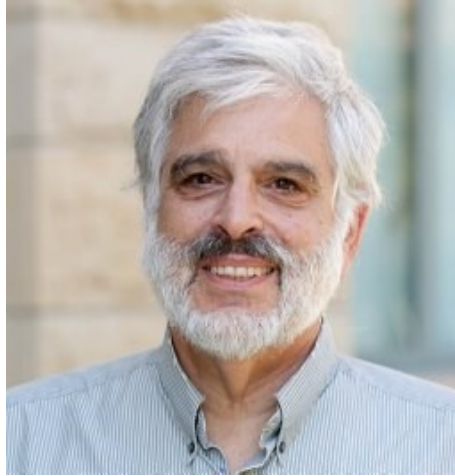
- **Database System Concepts** (7<sup>th</sup> Edition, 2019)
  - Abraham Silberschatz, Henry F. Korth, S. Sudarshan
- **Database systems: the complete book** (2<sup>nd</sup> Edition, 2014)
  - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
- **A first course in database systems** (3<sup>rd</sup> Edition, 2014)
  - Jeffrey D Ullman, Jennifer Widom
- **An Introduction to Database Systems** (8<sup>th</sup> Edition, 2004)
  - Christopher .J. Date
- **Fundamentals of Database Systems** (7<sup>th</sup> Edition, 2015)
  - Ramez Elmasri, Shamkant B. Navathe



# References (noted researcher of the field)



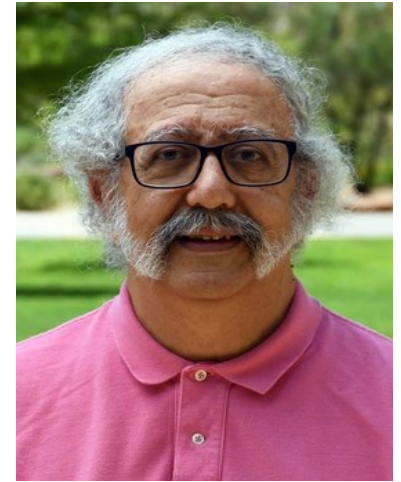
**Christopher J. Date**



**Jeffrey Ullman**



**Abraham Silberschatz**



**Ramez Elmasri**



# Grading policy

- Homework – individual works (4-5 points)
  - Writing assignments (WAs)
  - Programming assignments (PAs)
- Project – works in groups of **three** students (4 points)
- Class activities (1 point)
- Midterm exam – paper based/ practical exam (4-5 points)
- Final exam – paper based (6 points)
- **Additional activities** -- works in groups of **three** students ( $[-1, 3]$  points)
  - Research work with in-class presentations
    - We have time for three in class presentation
  - Homework design
  - Course material design



# View of Data

- A **database system** is a collection of interrelated data and a set of programs that allow users to access and modify these data.
- A major purpose of a database system is to provide users with an abstract view of the data.
  - Data models
    - A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
  - Data abstraction
    - Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.



# Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model



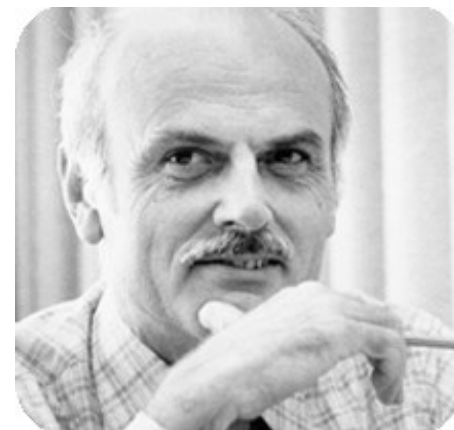
# Relational Model

- All the data is stored in various tables.
- Example of tabular data in the relational model

Columns

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

Rows



**Ted Codd**  
Turing Award 1981

(a) The *instructor* table



# A Sample Relational Database

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

(a) The *instructor* table

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table





# Levels of Abstraction

- **Physical level:** describes how a record (e.g., instructor) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

**type** *instructor* = **record**

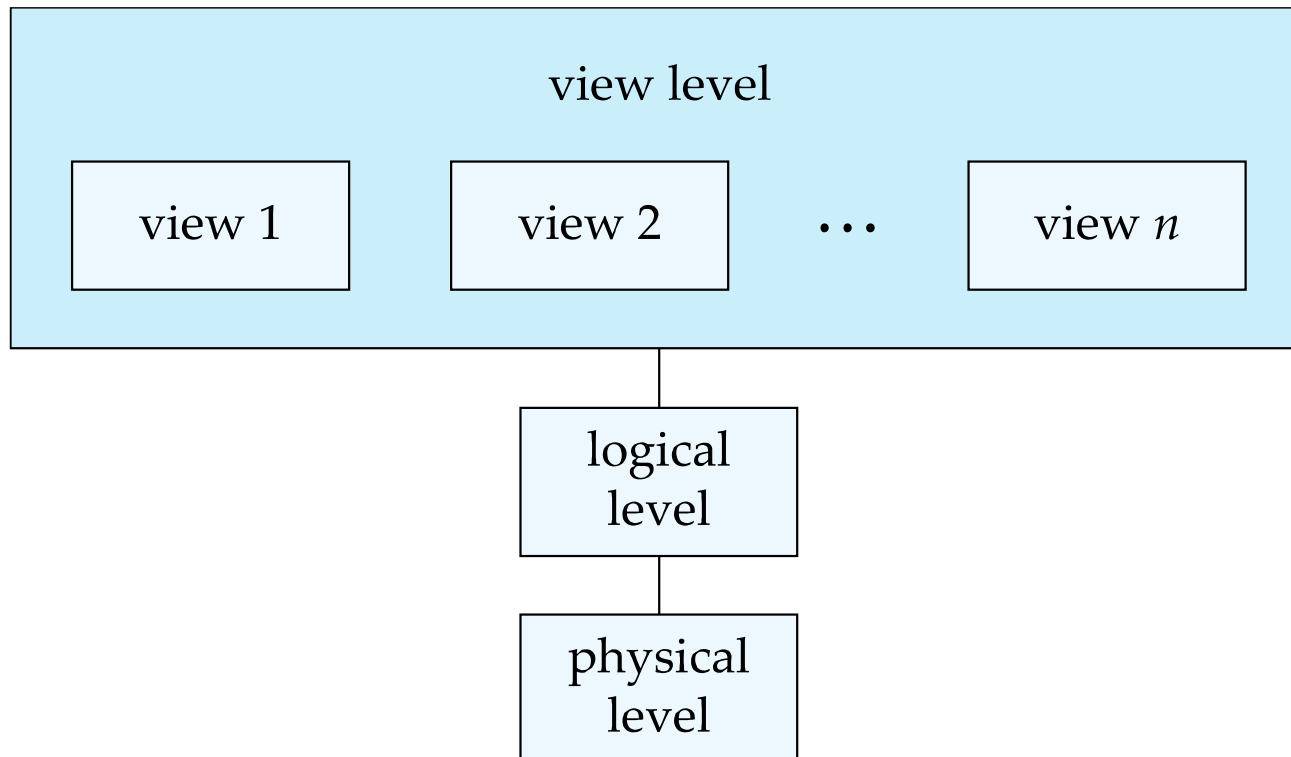
```
ID : string;  
name : string;  
dept_name : string;  
salary : integer;  
end;
```

- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.



# Database System Architecture

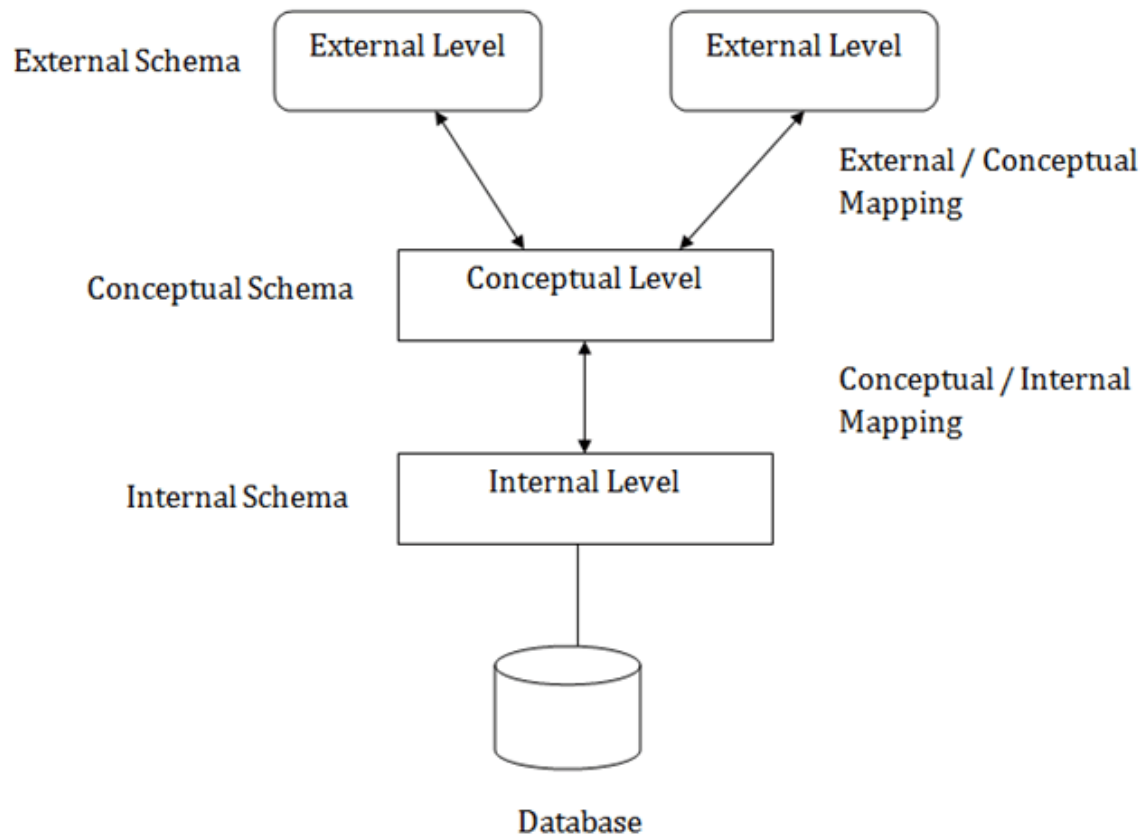
An architecture for a database system





# Three Schema Architecture

- The three schema architecture is also called **ANSI/SPARC architecture** or **three-level architecture**.
- This framework is used to describe the structure of a specific database system.





# Instances and Schemas

- Similar to types and variables in programming languages
- **Logical Schema** – the overall logical structure of the database
  - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - Analogous to type information of a variable in a program
- **Physical schema** – the overall physical structure of the database
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable



# Physical Data Independence

- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.



# Data Definition Language (DDL)

- Specification notation for defining the database schema

Example:      **create table** *instructor* (  
                         *ID*                **char**(5),  
                         *name*            **varchar**(20),  
                         *dept\_name* **varchar**(20),  
                         *salary*        **numeric**(8,2))

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what



# Data Manipulation Language (DML)

- Language for accessing and updating the data organized by the appropriate data model
  - DML also known as query language
- There are basically two types of data-manipulation language
  - **Procedural DML** -- require a user to specify what data are needed and how to get those data.
  - **Declarative DML** -- require a user to specify what data are needed without specifying how to get those data.
- Declarative DMLs are usually easier to learn and use than are procedural DMLs.
- Declarative DMLs are also referred to as non-procedural DMLs
- The portion of a DML that involves information retrieval is called a **query** language.



# SQL Query Language

- SQL query language is nonprocedural. A query takes as input several tables (possibly only one) and always returns a single table.

- Example to find all instructors in Comp. Sci. dept

```
select name  
from instructor  
where dept_name = 'Comp. Sci.'
```

- SQL is **NOT** a Turing machine equivalent language
- To be able to compute complex functions SQL is usually embedded in some higher-level language
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database





# Database Access from Application Program

- Non-procedural (Declarative) query languages such as SQL are not as powerful as a universal Turing machine.
- SQL does not support actions such as input from users, output to displays, or communication over the network.
- Such computations and actions must be written in a **host language**, such as C/C++, Java or Python, with embedded SQL queries that access the data in the database.
- **Application programs** -- are programs that are used to interact with the database in this fashion.



# Database Design

The process of designing the general structure of the database:

- Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
  - **Business decision** – What attributes should we record in the database?
  - **Computer Science decision** – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database



# 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.



# Storage Manager

- 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 (Cont.)

- 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.



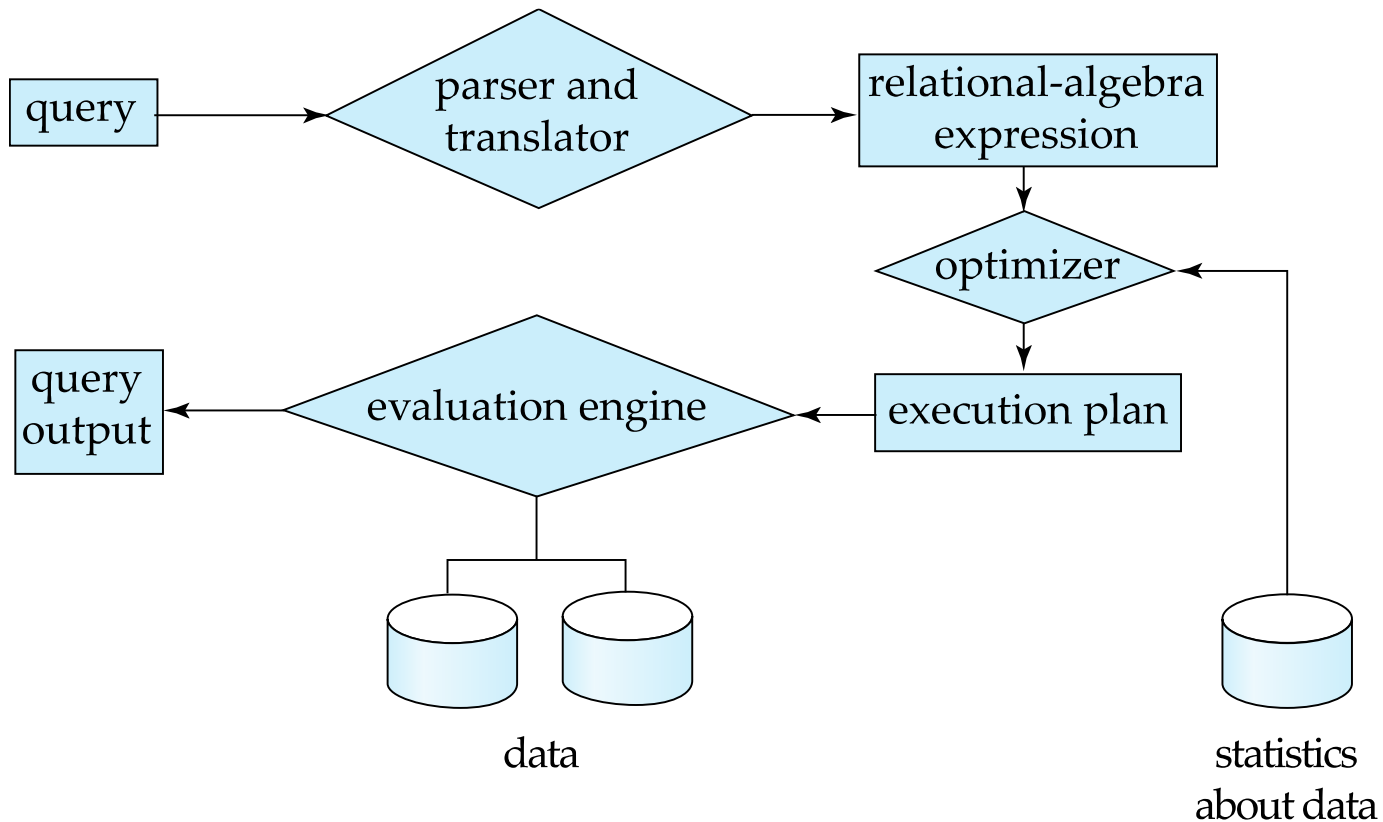
# Query Processor

- The query processor components include:
  - **DDL interpreter** -- interprets DDL statements and records the definitions in the data dictionary.
  - **DML compiler** -- translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan from among the various alternatives.
  - **Query evaluation engine** -- executes low-level instructions generated by the DML compiler.



# Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation





# 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.





# Transaction Concept

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.
- E.g., 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)
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions



# 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.



# Example of Fund Transfer (Cont.)

- **Consistency requirement** in above 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



# Example of Fund Transfer (Cont.)

- **Isolation requirement** — if between steps 3 and 6, another transaction T2 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).

**T1**

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

**T2**

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.



# 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.

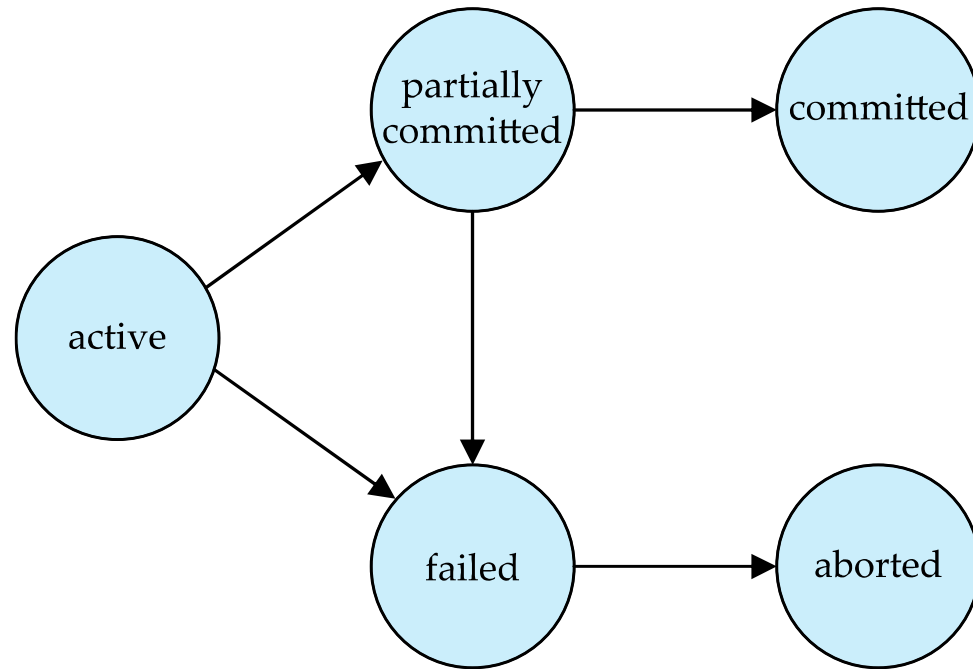


# Transaction State

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - Restart the transaction
    - Can be done only if no internal logical error
  - Kill the transaction
- **Committed** – after successful completion.



# Transaction State (Cont.)





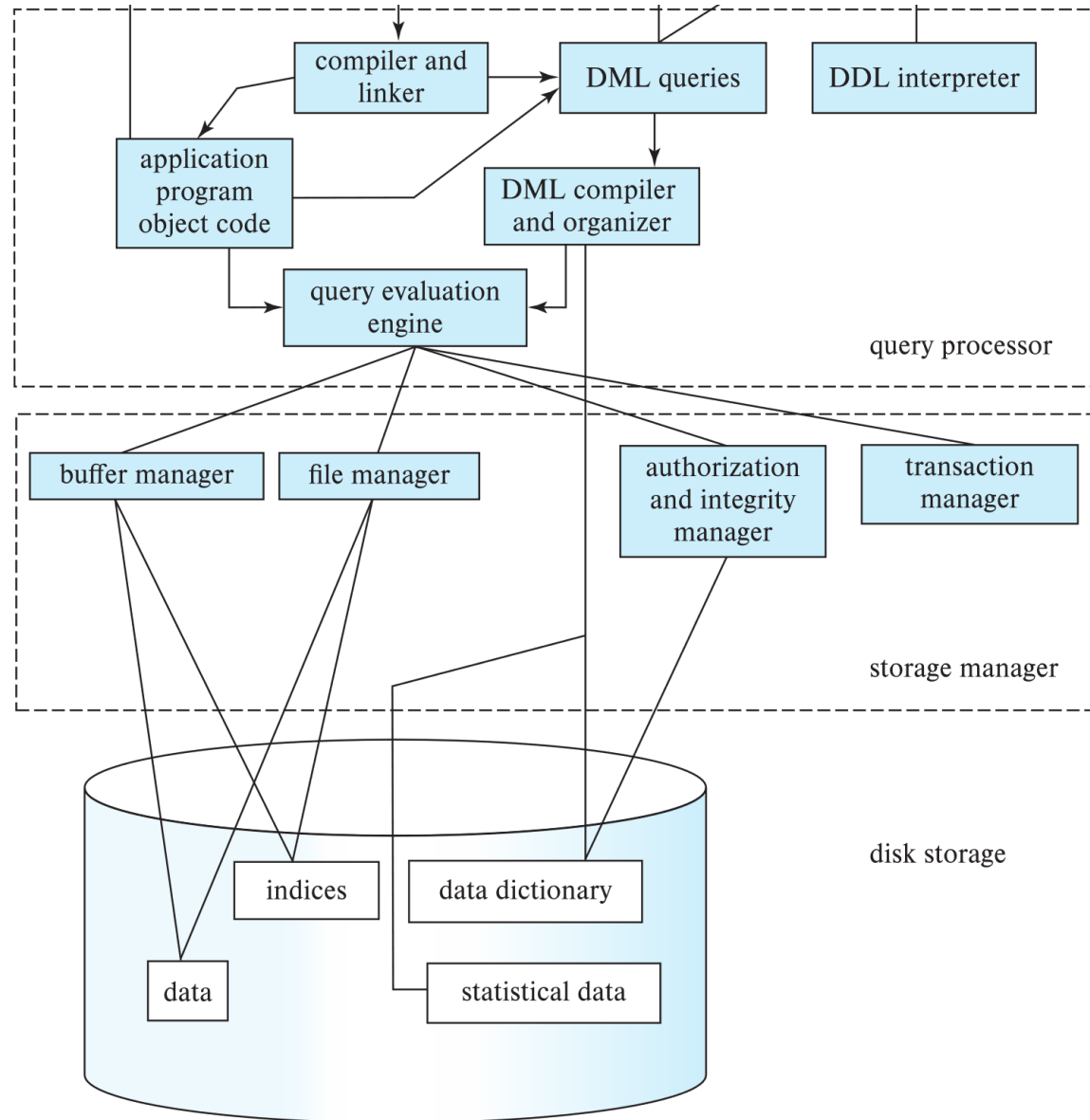
# Database Architectural Patterns

- Centralized databases
  - One to a few cores, shared memory
- Client-server,
  - One server machine executes work on behalf of multiple client machines.
- Parallel databases
  - Many core shared memory
  - Shared disk
  - Shared nothing
- Distributed databases
  - Geographical distribution
  - Schema/data heterogeneity





# Database Architecture (Centralized/Shared-Memory)





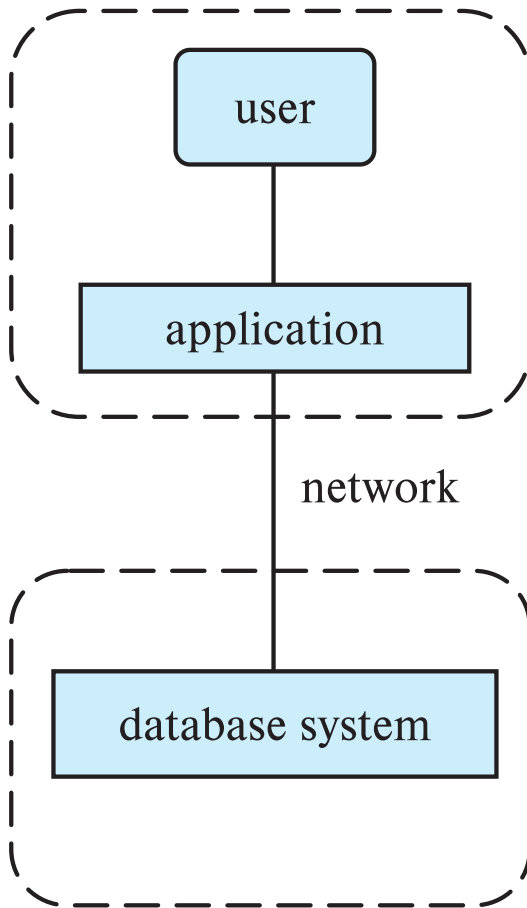
# Database Applications

Database applications are usually partitioned into two or three parts

- **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.



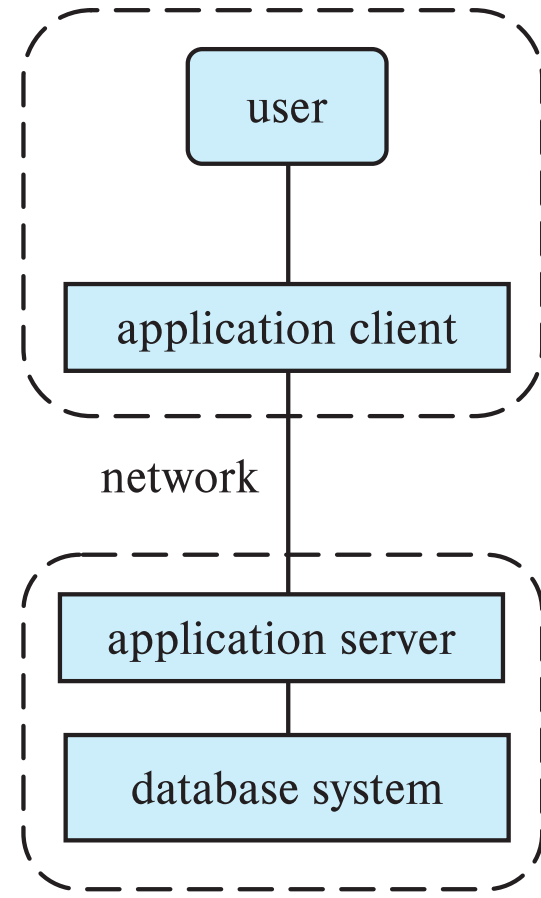
# Two-tier and three-tier architectures



(a) Two-tier architecture

client

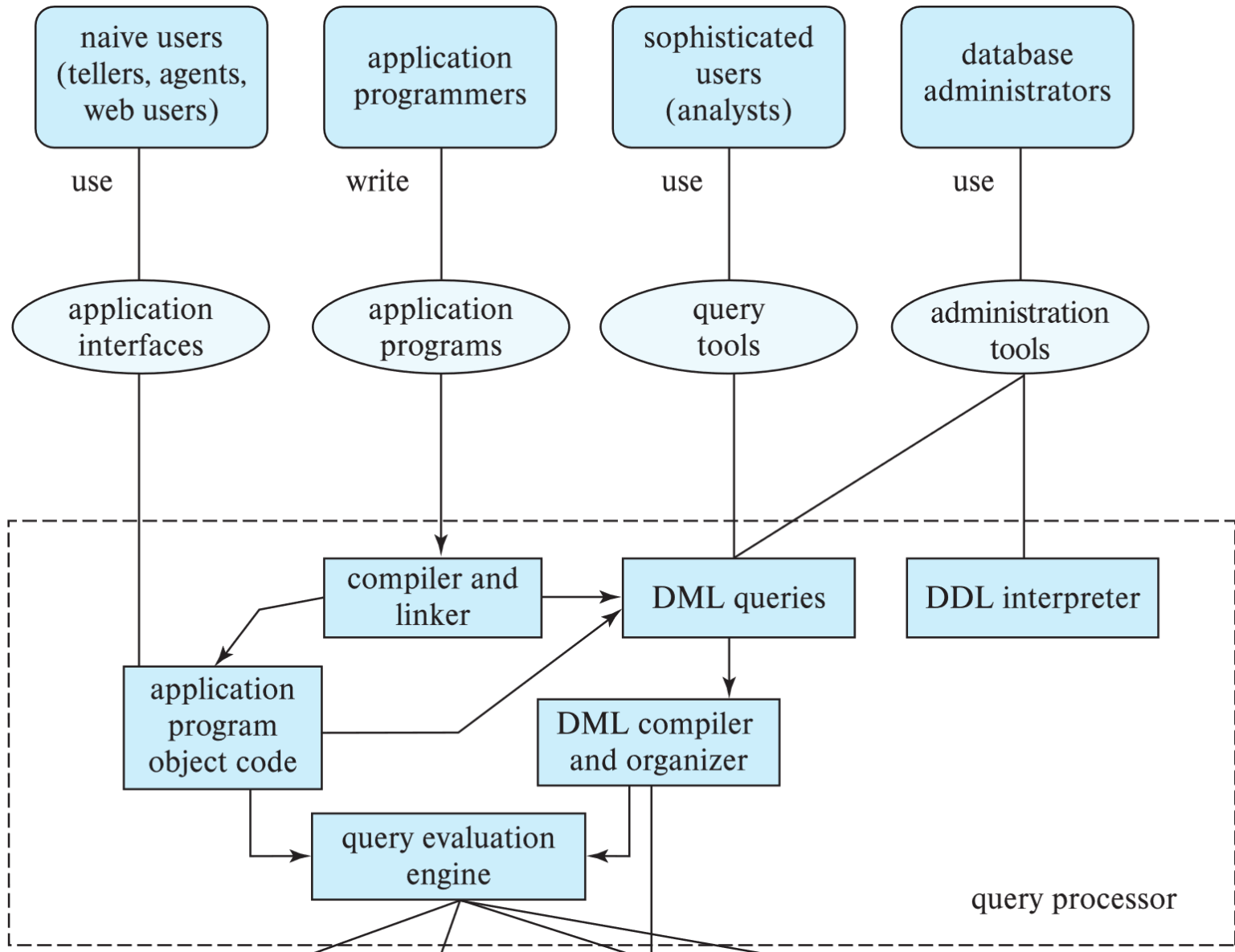
server



(b) Three-tier architecture



# 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



# History of Database Systems

- **1950s and early 1960s:**
  - Data processing using magnetic tapes for storage
    - Tapes provided only sequential access
  - Punched cards for input
- **Late 1960s and 1970s:**
  - Hard disks allowed direct access to data
  - Network and hierarchical data models in widespread use
  - **Ted Codd** defines the **relational data model**
    - Would win the ACM Turing Award for this work
    - IBM Research begins System R prototype
    - UC Berkeley (Michael Stonebraker) begins Ingres prototype
    - Oracle releases first commercial relational database
  - High-performance (for the era) transaction processing



# History of Database Systems (Cont.)

- 1980s:
  - Research relational prototypes evolve into commercial systems
    - SQL becomes industrial standard
  - Parallel and distributed database systems
    - Wisconsin, IBM, Teradata
  - Object-oriented database systems
- 1990s:
  - Large decision support and data-mining applications
  - Large multi-terabyte data warehouses
  - Emergence of Web commerce



# History of Database Systems (Cont.)

- 2000s
  - Big data storage systems
    - Google BigTable, Yahoo PNuts, Amazon,
    - “**NoSQL**” systems.
  - Big data analysis: beyond SQL
    - Map reduce and friends
- 2010s
  - SQL reloaded
    - SQL front end to Map Reduce systems
    - Massively parallel database systems
    - Multi-core main-memory databases





# OLTP and OLAP

- **OLTP** stands for **O**nline **T**ransactional **P**rocessing
  - The main focus of **OLTP** is to excel in looking up filtered information and manage current **C**reate, **R**ead, **U**ppdate, and **D**eletion while a transaction (**CRUD operations**)
  - **IBM DB2, Oracle, MySQL, Microsoft SQL, Postgresql**
- **OLAP** stands for **O**nline **A**nalytical **P**rocessing system.
  - The main focus of OLAP to help us extract useful information from different multi-dimensional data.
  - **IBM Netezza, Amazon Redshift, HP Vertica**



# OLTP and OLAP

	OLTP	OLAP
Function	Day to day operation	Decision support
Database Design	Application oriented	Subject oriented
Data	Current, up-to-date detailed, flat relational, isolated	Historical, summarized multi-dimensional, consolidated
Usage	Repetitive	Ad-hoc
Access	Read/Write	Lots of scans
Unit of Work	Short, simple transaction	Complex query
Database Size	Gigabytes	Terabytes
Metric	Transaction throughput	Query throughput, response



# End of Chapter 1