Topic 1: Introduction – Basic Concepts of DBMS (Ch. 1)

Database System Concepts

(Modified for CS 4513)

Outline

- Database applications and purposes of database systems
  - Database Management Systems (DBMS); Drawbacks of using file systems to store data
- Abstraction and data models
  - Levels of abstraction; Views of data; Instances and schemas; Data models
- Relational model and database languages
  - Relational model; Data definition language, SQL
- Database design
  - Database design; Entity Relationship Model
- Database system internals and architectures
  - Storage management; Query processing; Transaction management; DBA's responsibilities; Database system internals; Database system architectures
- Database system evolution
  - History of database systems; When not to use a database system

Database Management System (DBMS)

- 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 Applications:
  - Banking: transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Online retailers: order tracking, customized recommendations
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases can be very large.
- Databases touch all aspects of our lives

University Database Example

- 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
- In the early days, database applications were built directly on top of file systems

Drawbacks of using file systems to store data

- Data redundancy and inconsistency
  - Multiple file formats, 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., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

Drawbacks of using file systems to store data (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
    - Example: 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

Levels of Abstraction

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

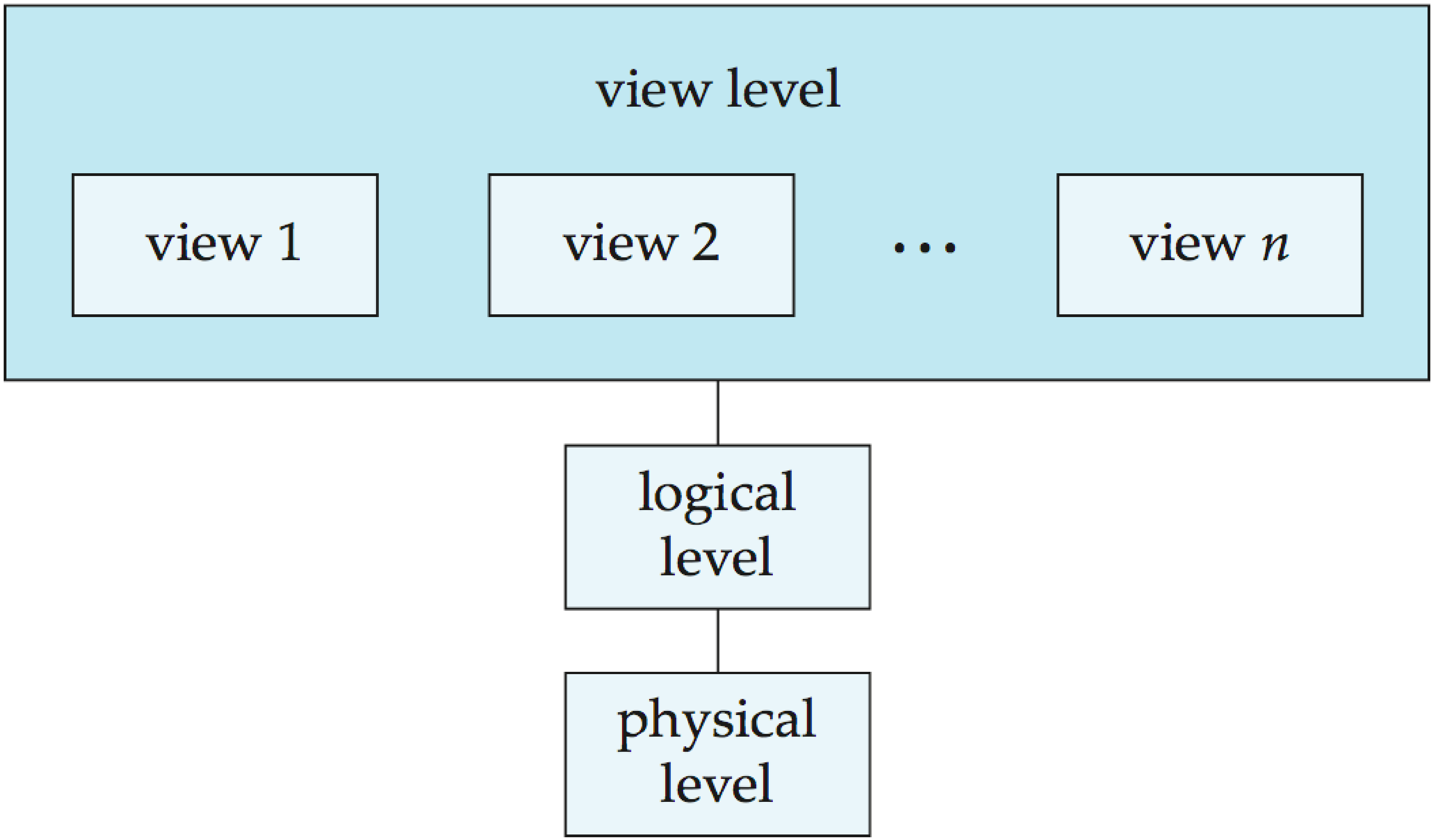
  type *instructor* = **record**

  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.

View of Data



Instances and Schemas

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database
  - Example: The database consists of information about a set of customers and accounts and the relationship between them
  - Analogous to type information of a variable in a program
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
- **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 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)
- Semistructured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Relational Model

- Relational model (Chapter 2)
- Example of tabular data in the relational model

Columns

Rows

ID	*name*	*dept_name*	*salary*
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

A Sample Relational Database

ID	*name*	*dept_name*	*salary*
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

dept_name	*building*	*budget*
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

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)
    - Referential integrity (**references** constraint in SQL)
      - e.g. *dept_name* value in any *instructor* tuple must appear in *department* relation
  - Authorization

SQL

- **SQL**: widely used non-procedural language
  - Example: Find the name of the instructor with ID 22222

    select *name*
    from *instructor*
    where *instructor.ID* = ‘22222’
  - Example: Find the ID and building of instructors in the Physics dept.

    select *instructor*.*ID*, *department*.*building*
    from *instructor*, *department*
    where *instructor.dept_name* = *department.dept_name* **and**
    department.dept_name = ‘Physics’

- 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
- Chapters 3, 4 and 5

Database Design?

- Is there any problem with this design?

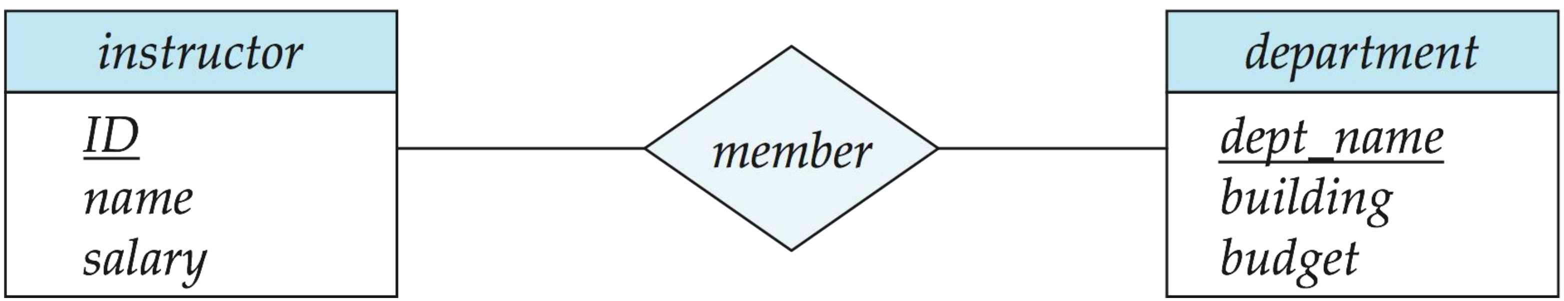
ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Design Approaches

- Normalization Theory (Chapter 7)
  - Formalize what designs are bad, and test for them
- Entity Relationship Model (Chapter 6)
  - Models an enterprise as a collection of *entities* and *relationships*
    - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects
      - Described by a set of *attributes*
    - Relationship: an association among several entities
  - Represented diagrammatically by an *entity-relationship diagram:*

The Entity-Relationship Model

- Models an enterprise as a collection of *entities* and *relationships*
  - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects
    - Described by a set of *attributes*
  - Relationship: an association among several entities
- Represented diagrammatically by an *entity-relationship diagram:*



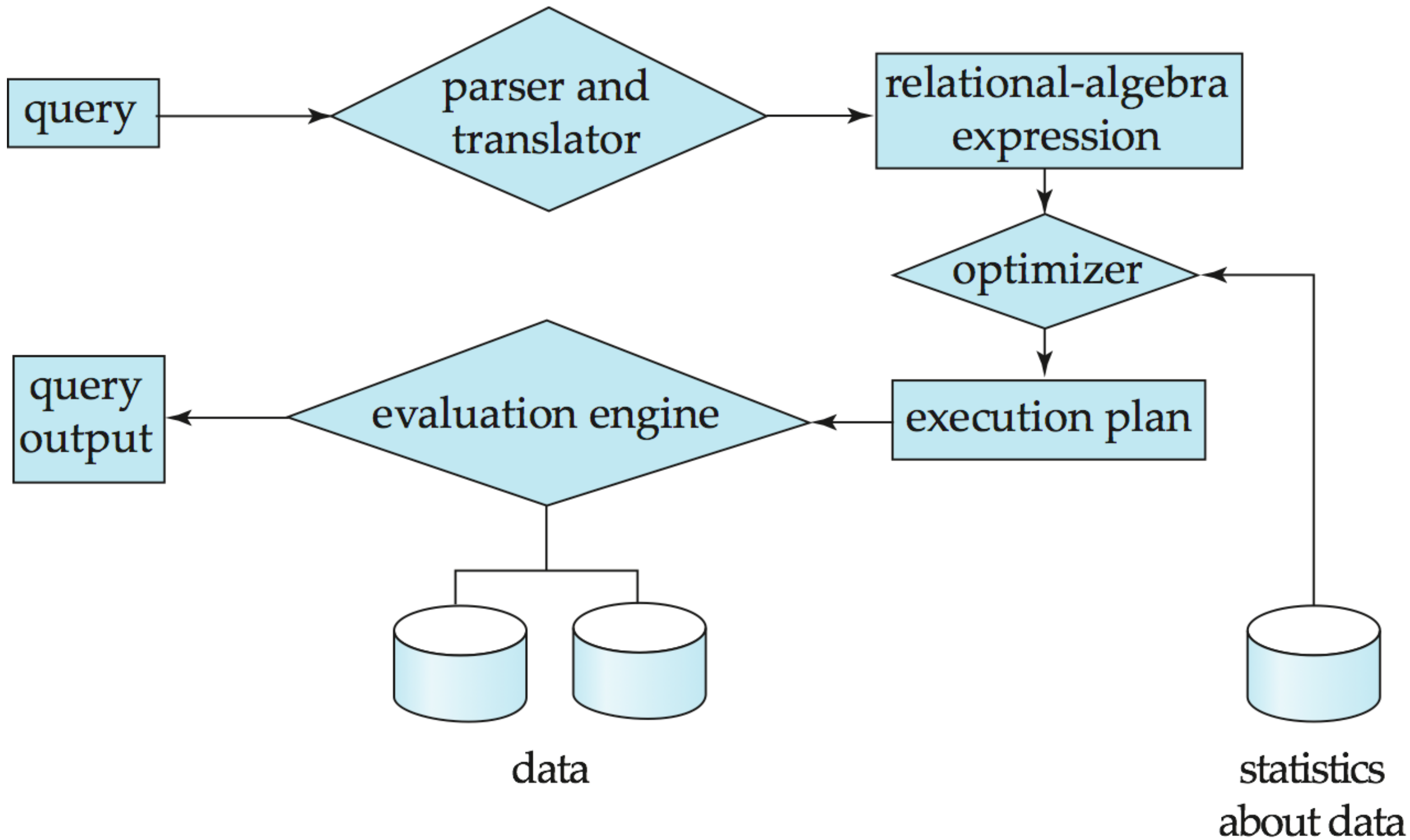
What happened to dept_name of instructor?

Storage Management

- **Storage manager** is 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 file manager
  - Efficient storing, retrieving and updating of data
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Transaction Management

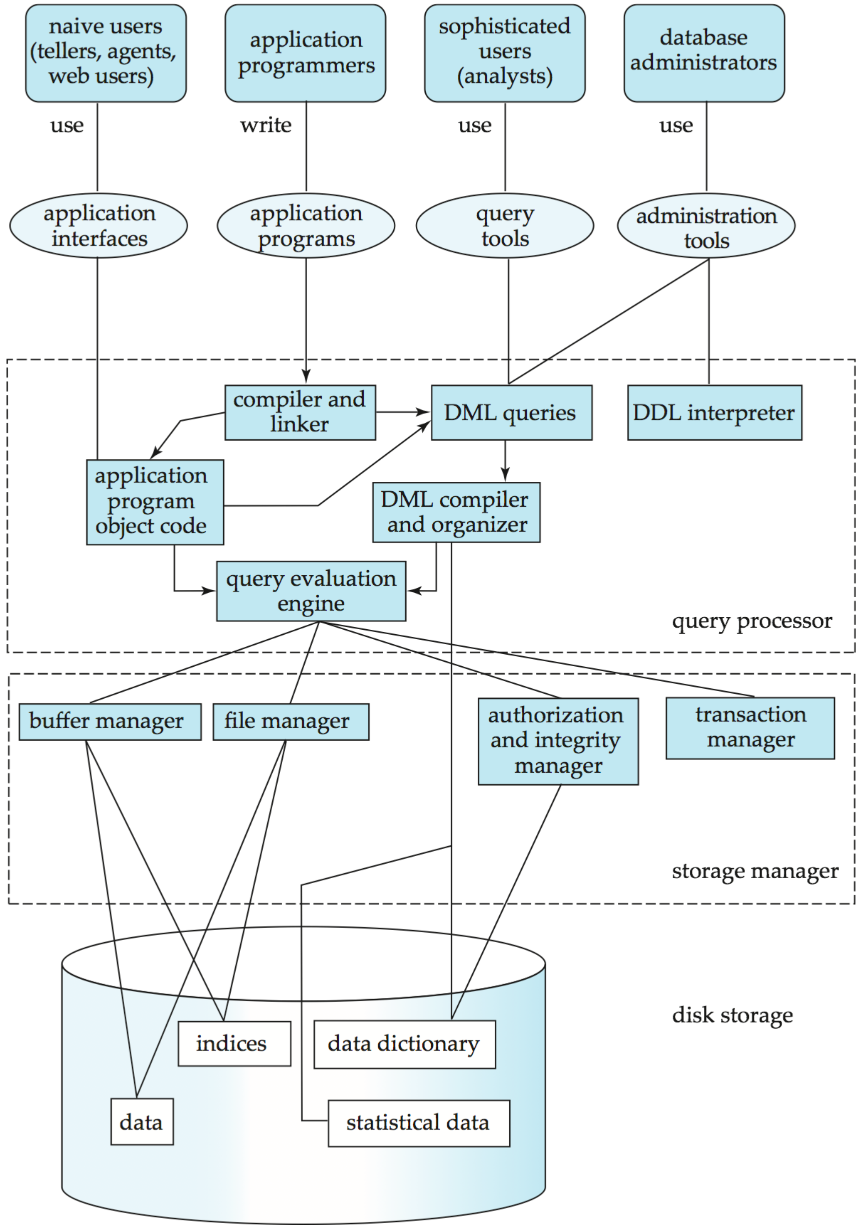
- What if the system fails?
- What if more than one user is concurrently updating the same data?
- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (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.

Database Administrator

A person who has central control over the system is called a **database administrator** (**DBA**), whose functions are:

- 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 and ensuring that performance is not degraded by very expensive tasks submitted by some users

Database System Internals

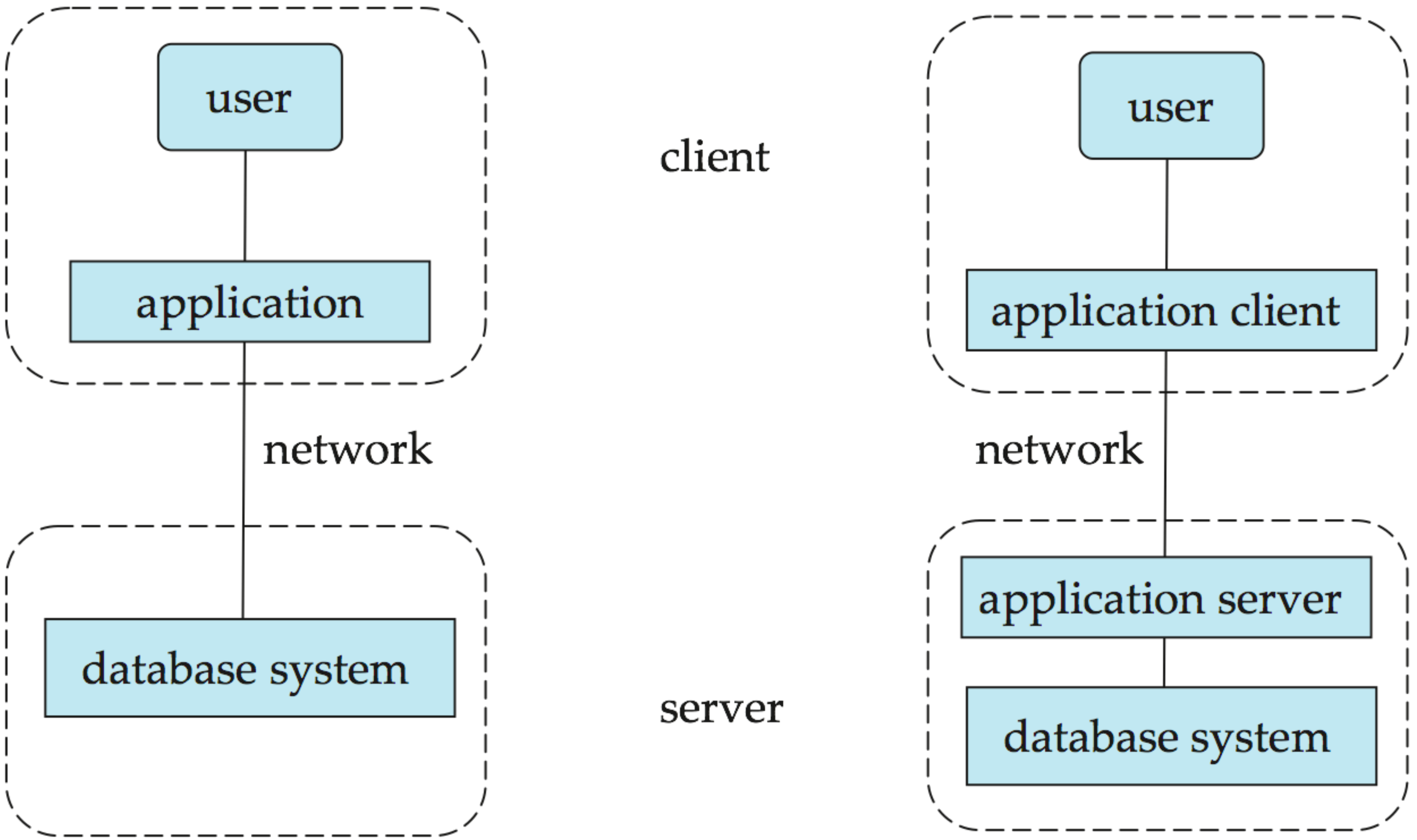


Database Architecture

The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

- Centralized
- Client-server
- Parallel (multi-processor)
- Distributed

Client-Server DB Architectures



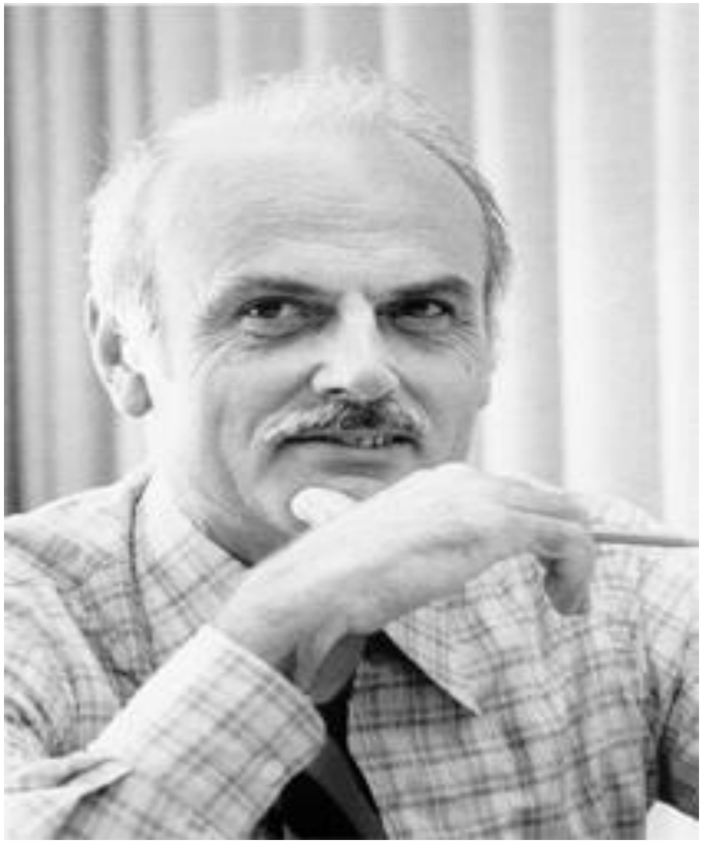
(a) Two-tier architecture

(b) Three-tier architecture

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
    - Won the ACM Turing Award in 1981 for this work
    - IBM Research begins System R prototype
    - UC Berkeley begins Ingres prototype
    - Oracle releases the first commercial relational database
  - High-performance transaction processing

Edgar Frank "Ted" Codd
(August 19, 1923 – April 18, 2003)
Tuning Award 1981



History (cont.)

- 1980s:
  - Research relational prototypes evolve into commercial systems
    - SQL becomes industrial standard
  - Parallel and distributed database systems
  - Object-oriented database systems
- 1990s:
  - Large decision support and data-mining applications
  - Large multi-terabyte data warehouses
  - Emergence of Web commerce
- 2000s:
  - XML and XQuery standards
  - Automated database administration
  - Big data storage systems
    - Google BigTable, Yahoo PNuts, Amazon
    - NoSQL systems
- 2010s:
  - SQL front end to Map Reduce Systems
  - Massively parallel database systems
  - Multi-core main-memory databases

When Not to Use a DBMS?

End of Topic 1