Part 1: Introduction

Database System Concepts, 7th Ed.

Structured and Unstructured Textual Information

- A Database Management System (DBMS) is a complex software system whose task is to manage a large, complex collection of primarily structured information
- Structured Information are organized in discrete units (entities)
- Entities of the same type are organized in some predefined ways:
  - They have the same number of attributes
  - Each attribute has
    - The same predefined format, e.g. the number of bytes required to store an attribute
- Unstructured information refers to data that
  - Does not have a rigid format
  - Does not distinguish information into specific items
  - Usually free text

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

- 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
- Navigation systems: For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.

Database Systems vs Information Retrieval Systems

- Information Retrieval (IR) or Document Retrieval is finding documents or objects that satisfy particular information requirements from large repositories which can contain textual or multimedia documents
  - Library Search
  - Image Search
  - Video Search
  - Music Search

Textual Retrieval is Well-Developed: term-document count matrices

- Count the number of occurrences of a term in a document
  (7 terms, 6 documents)

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

tf-idf Weighting

- The relevance of a document increases with the number of occurrences of the search term within the document
- Relevance increases with the rarity of the term in the collection
  - Frequent terms are less informative than rare terms (e.g. stop words such as “the”, “be”, “to”)
  - Rare terms are terms that do not occur in many documents
- The *tf-idf* weight of a term is the product of its *tf* weight (term frequency) and its *idf* weight (inverse document frequency)

$$\text{tf-idf}_{t,d} = (1 + \log_{10} \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

  - N is the number of documents in the collection
  - $tf_{t,d}$ is the number of times term t occurs in document d
  - df_t is the document frequency of t
    - the number of documents in the collection that contains term t

tf-idf Weighting

Each document is now represented by a real-valued vector of tf-idf weights $\in R^{|V|}$. Here, V=7, which is the number of terms.

Note: this example is not based on the data in Slide 8.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

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., 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

- 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 the above problems

University Database Example

- 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

View of Data

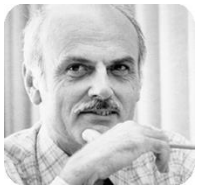
- A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data
- A major purpose of a database system is to provide users with an abstract view of the data
  - Data models
    - A collection of conceptual tools for describing data, data 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 (JSON, 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



Ted Codd
Turing Award 1981

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

Levels of Abstraction

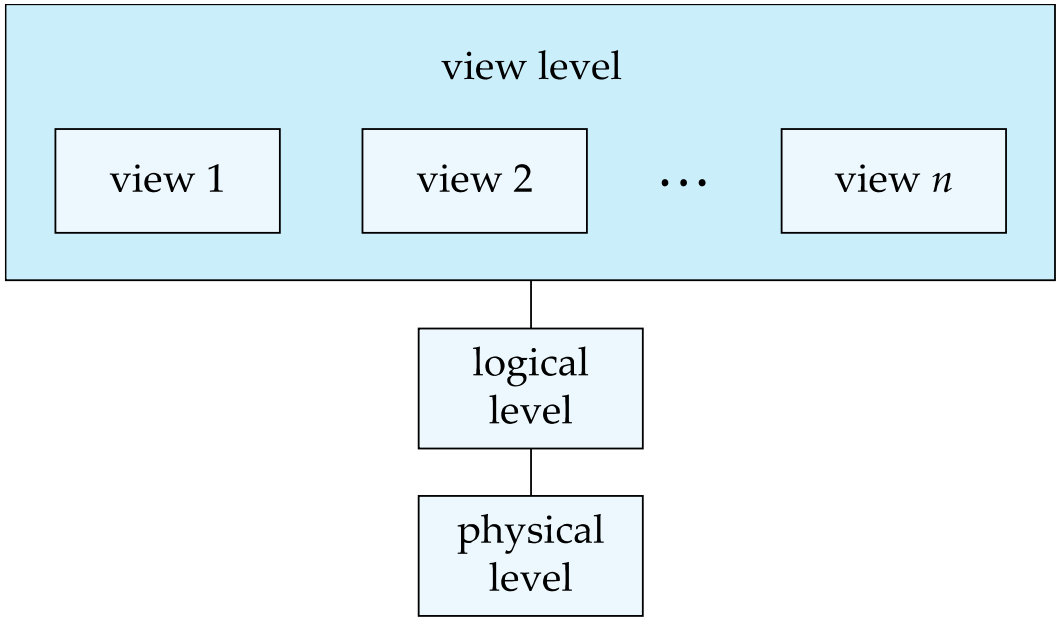
- **Physical level**: describes how a record (e.g., instructor) is stored
- **Logical level**: describes data stored in database, and the relationships among the data

```
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

View of Data

An architecture for a database system



Instances and Schemas

- **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 extraction is called a **query** language

SQL (Structured Query Language)

- SQL 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
  - some computations that are possible using a general-purpose programming language but are not possible using SQL
- To be able to compute complex functions SQL is usually embedded in some higher-level language

Database Access from Application Program

- Non-procedural query languages such as SQL are not as powerful as a universal Turing machine
  - **Church–Turing thesis** or **Church–Turing conjecture** is a hypothesis about the nature of computable functions, which states that a function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine
    - **effective computability** means computable by a Turing machine
  - The **halting problem** is an **undecidable problem**: there is no algorithm that correctly determines whether arbitrary programs eventually halt when run
- 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

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 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 Components

- 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 component
  - The query processor component
  - The transaction management component

Storage Manager

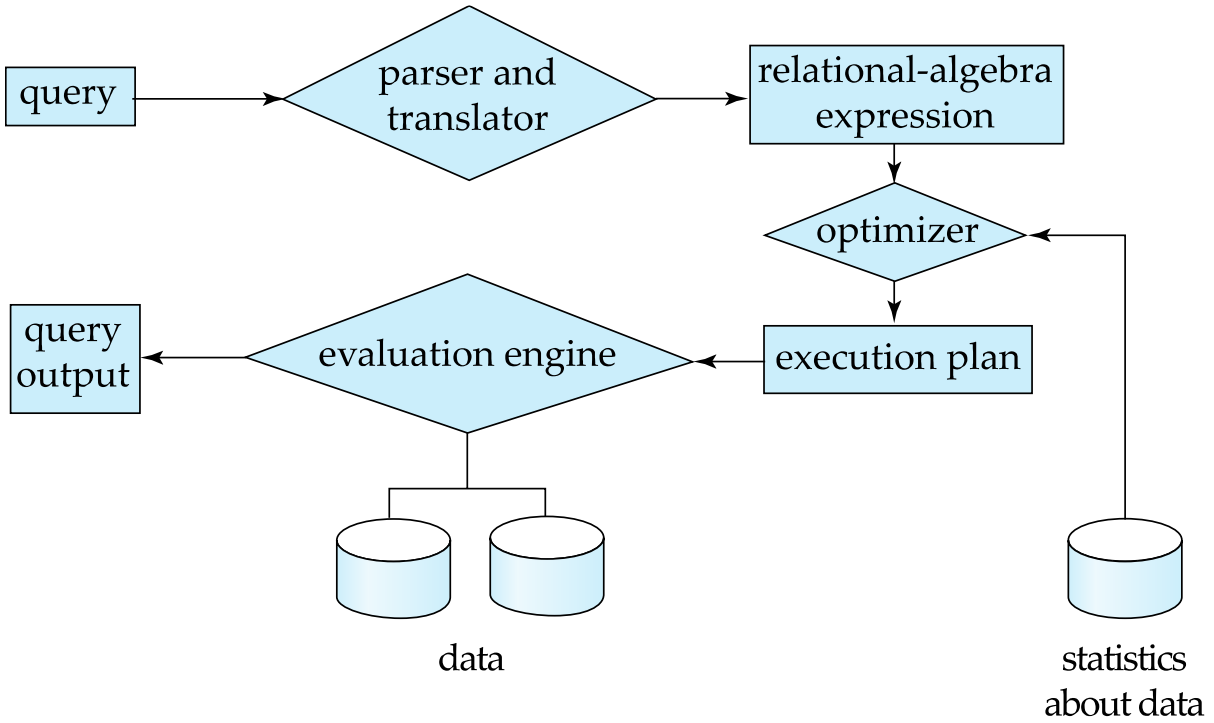
- A program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system
- The storage manager is responsible to the following tasks:
  - Interaction with the OS file manager
  - Efficient storing, retrieving and updating of data
- The storage manager 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
  - The SABRE System processes nearly 20,000 transactions/second
- **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
  - ACID (Atomicity, Consistency, Isolation, Durability) properties
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database

Database Architecture

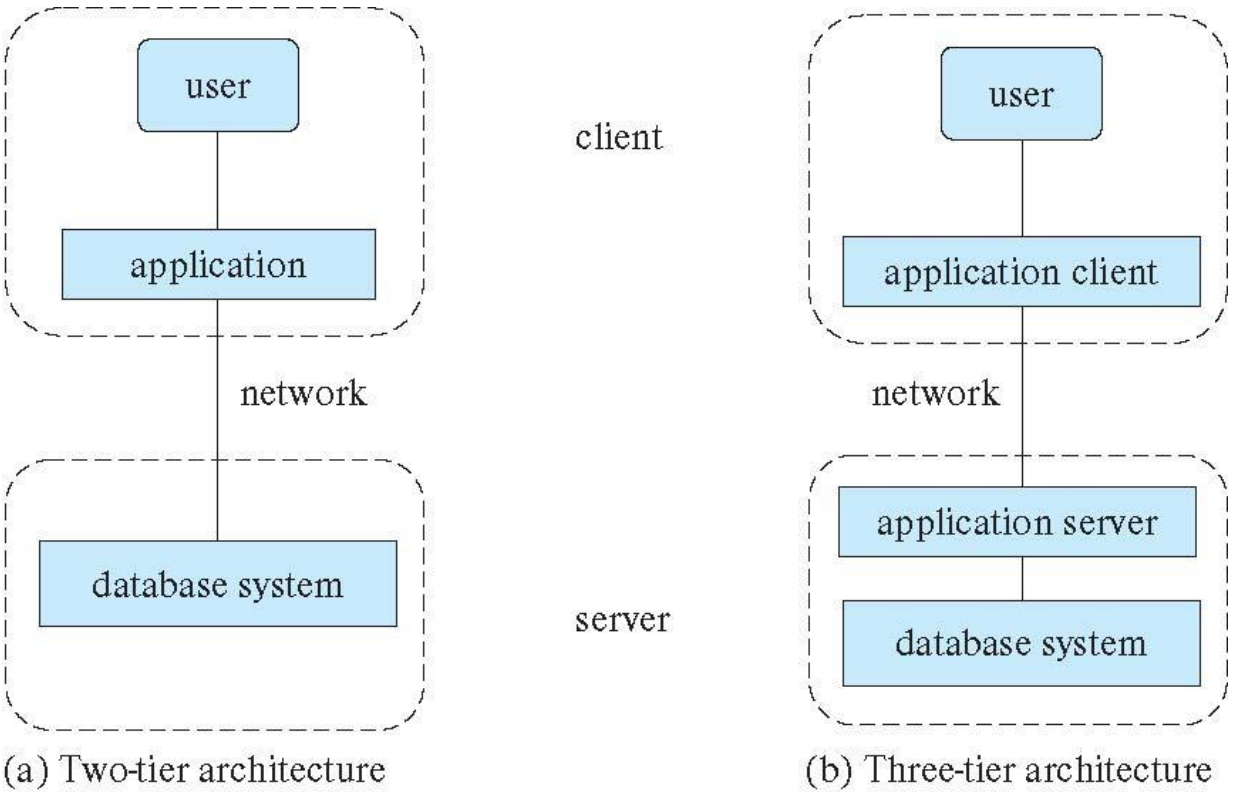
- Centralized databases
  - A few cores, shared memory
- Client-server
  - One server machine executes work on behalf of multiple client machines
- Parallel databases
  - Many cores, shared memory
  - Shared disk
  - Shared nothing
- Distributed databases
  - Geographical distribution
  - Schema/data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

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
  - The application server in turn communicates with a database system to access data

Two-tier and three-tier architectures



(a) Two-tier architecture

(b) Three-tier architecture

Database Users

There are different types of database-system users

- Naive users -- unsophisticated users who interact with the system by invoking one of the application programs that have been written previously
- Application programmers -- who write programs that interacts with the database
- Sophisticated users -- interact with the system without writing programs
  - using a database query language or by
  - using tools such as data analysis software

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

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

Michael Stonebraker
Turing Award 2014

History of Database Systems

- 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

History of Database Systems

- 2000s
  - Big data storage systems and Analysis
    - “NoSQL” systems
    - Map reduce
- 2010s
    - Massively parallel database systems
    - Multi-core main-memory databases