Chapter 1: Introduction

Database System Concepts, 6th Ed.

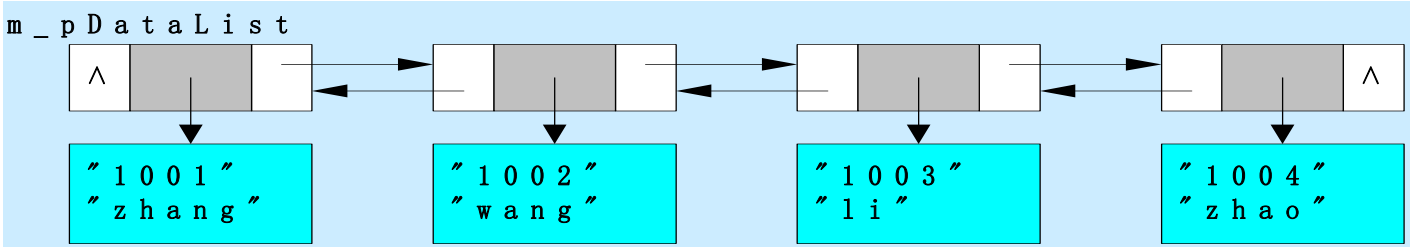
Outline

- The Need for Databases
- Data Models
- Relational Databases
- Database Design
- Storage Manager
- Query Processing
- Transaction Manager

A Motivating Example

- Suppose we are building a system to store the information about:
  - students
  - courses
  - professors
  - who takes what, who teaches what

Definition: CTypedPtrList <CObList, CStudent*> m_pDataList;



Application Requirements

- **store the data for a long period of time**
  - large amounts (100s of GB)
  - protect against crashes
  - protect against unauthorized use
- **allow users to query/update:**
  - who teaches “CS 173”
  - insert “Mary” in “CS 311”
- **allow several (100s, 1000s) users to access the data simultaneously**
- **allow administrators to change the schema**
  - add information about TAs

Trying Without a DBMS

- **Why Direct Implementation Won't Work:**
- **Storing data: file system is limited**
  - size less than 4GB (on 32 bits machines)
  - when system crashes we may lose data
  - password-based authorization insufficient
- **Query/update:**
  - need to write a new C++/Java program for every new query
  - need to worry about performance

Trying Without a DBMS(Cont)

- Concurrency: limited protection
  - need to worry about interfering with other users
  - need to offer different views to different users (e.g. registrar, students, professors)
- Schema change:
  - entails changing file formats
  - need to rewrite virtually all applications
- consistency：
  - update several files with same data
- Better let a database system handle it

Database Management System (DBMS)

- DBMS contains the information about a particular enterprise/organization
  - 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,hard to retain the interrelation
- **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

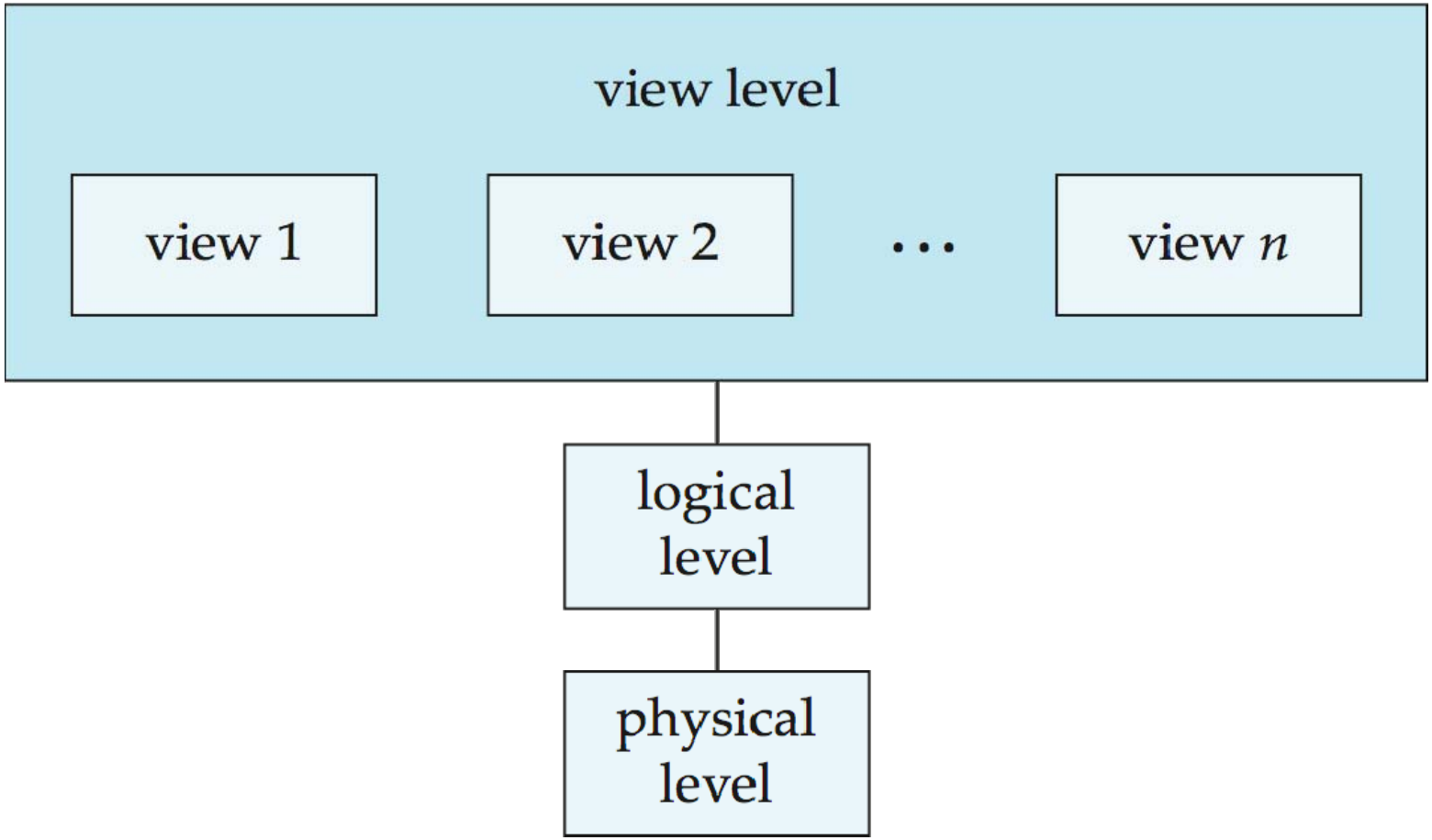
- **Physical level:** describes how a record (e.g., instructor) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

```
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

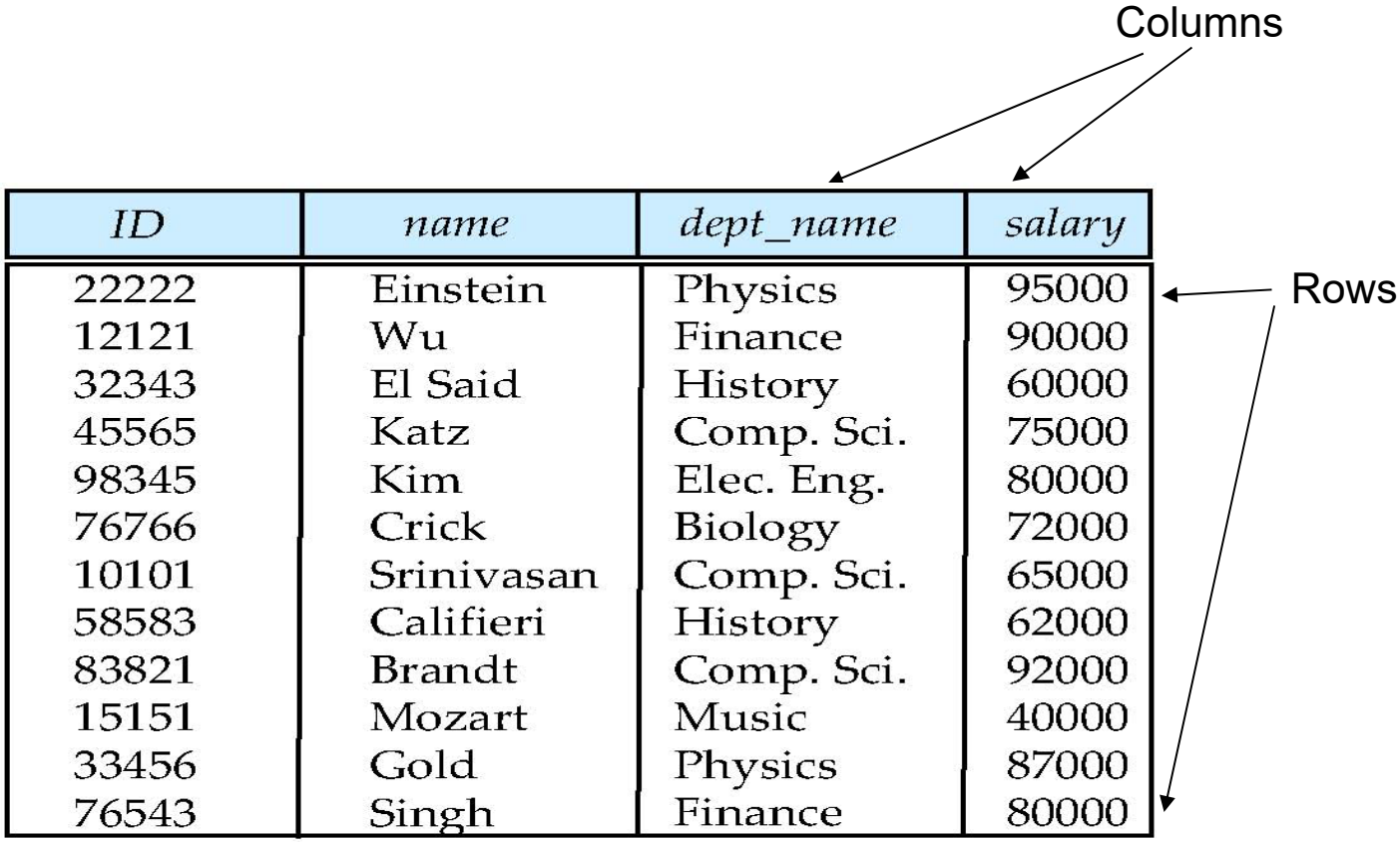
- Similar to types and variables in programming languages
- **Logical Schema** – the overall **logical structure** of the database
  - *Example:* The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - Analogous to variable declaration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

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



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

Database Languages

- DML(Data manipulation language) expresses database queries and updates;
- DDL(Data definition language) specifies the database schema.
- They are not two separated languages, they are usually parts of a single database language.

Data Manipulation Language (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as ***query language***
- Two classes of languages
  - **Pure** – used for proving properties about computational power and for **optimization**
    - Relational Algebra
    - Tuple relational calculus
    - Domain relational calculus
  - **Commercial** – used in commercial systems
    - SQL is the most widely used commercial language

SQL

- The most widely used **commercial DML** language
- **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 the methods:
  - Language extensions to allow embedded SQL
  - Application program **interface** (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

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

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 Design (Cont.)

- Is there any problem with this r

Can we add one department information into this table directly?

ID	name	salary	dept_name	buildi	
22222	Einstein	95000	Physics	Watso	
12121	Wu	90000	Finance	Painter	20000
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

- Need to come up with a methodology to ensure that each of the relations in the database is "**good**"
- Two ways of doing so:
  - Entity Relationship Model (Chapter 7)
    - Models an enterprise as a collection of *entities* and *relationships*
    - Represented diagrammatically by an *entity-relationship diagram:*
  - Normalization Theory (Chapter 8)
    - Formalize what designs are bad, and test for them

Object-Relational Data Models

- Relational model: flat, “atomic” values
- Object Relational Data Models
  - **Extend** the relational data model by including object orientation and constructs to deal with added data types.
  - Allow attributes of tuples to have **complex types,** including non-atomic values such as nested relations.
  - Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
  - Provide upward compatibility with existing relational languages.

XML: Extensible Markup Language

- Defined by the WWW Consortium (W3C)
- Originally intended as a **document markup language** not a database language
- The ability to specify new tags, and to create nested tag structures made XML a great way to **exchange data**, not just documents
- XML has become the basis for all new generation data **interchange formats.**
- A wide variety of tools is available for **parsing, browsing and querying** XML documents/data

Database Engine

There are 3 main components in a database management system.

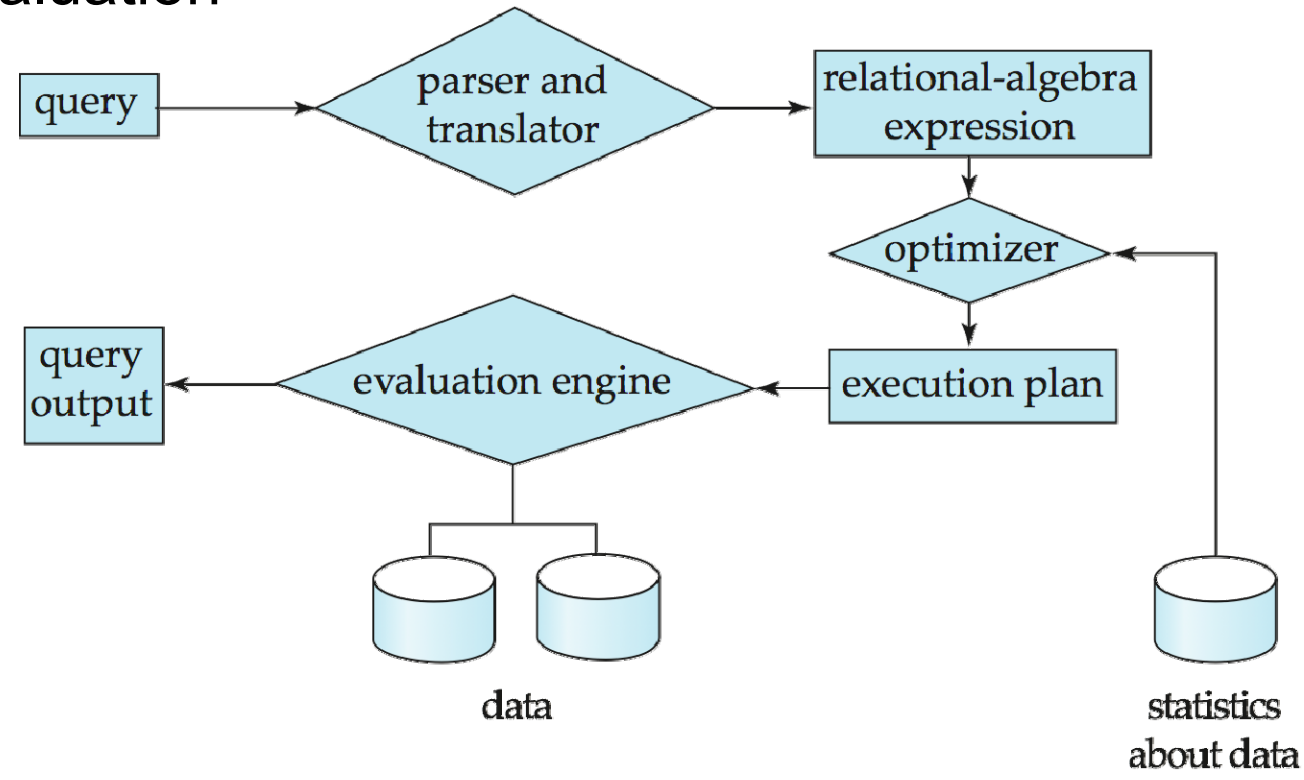
- Storage manager
- Query processing
- Transaction manager

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 OS file manager
  - storing, retrieving and updating of data (**efficient)**
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



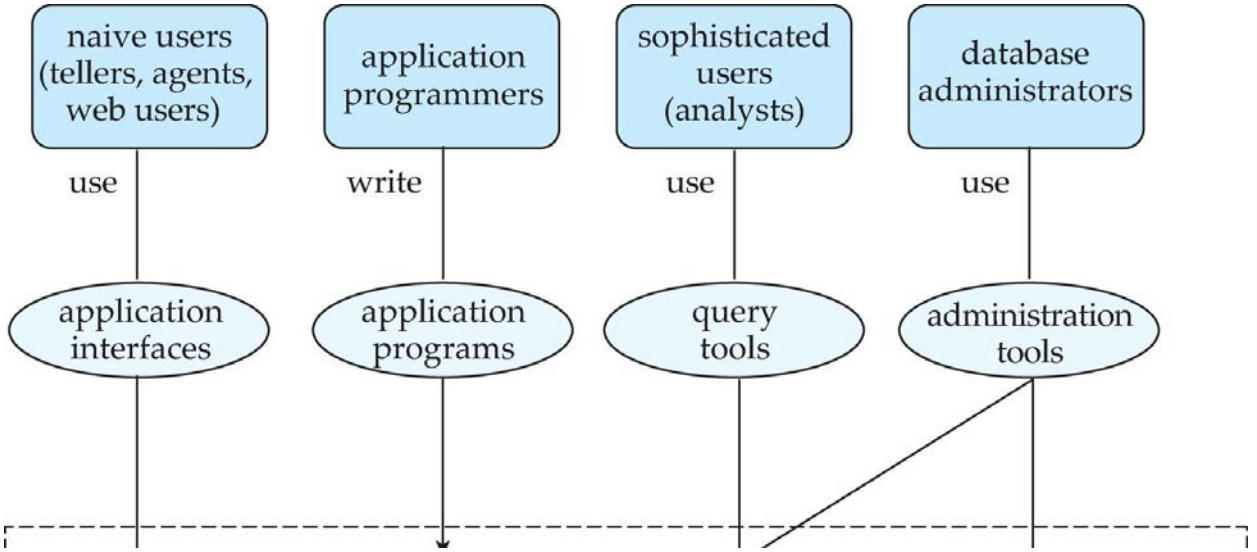
Query Processing (Cont.)

- Alternative ways of evaluating a given query
  - **Equivalent** expressions
  - Different algorithms for each operation
- **Cost difference** between a good and a bad way of evaluating a query can be **enormous**
- Need to estimate the cost of operations
  - Depends critically *on statistical information* about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

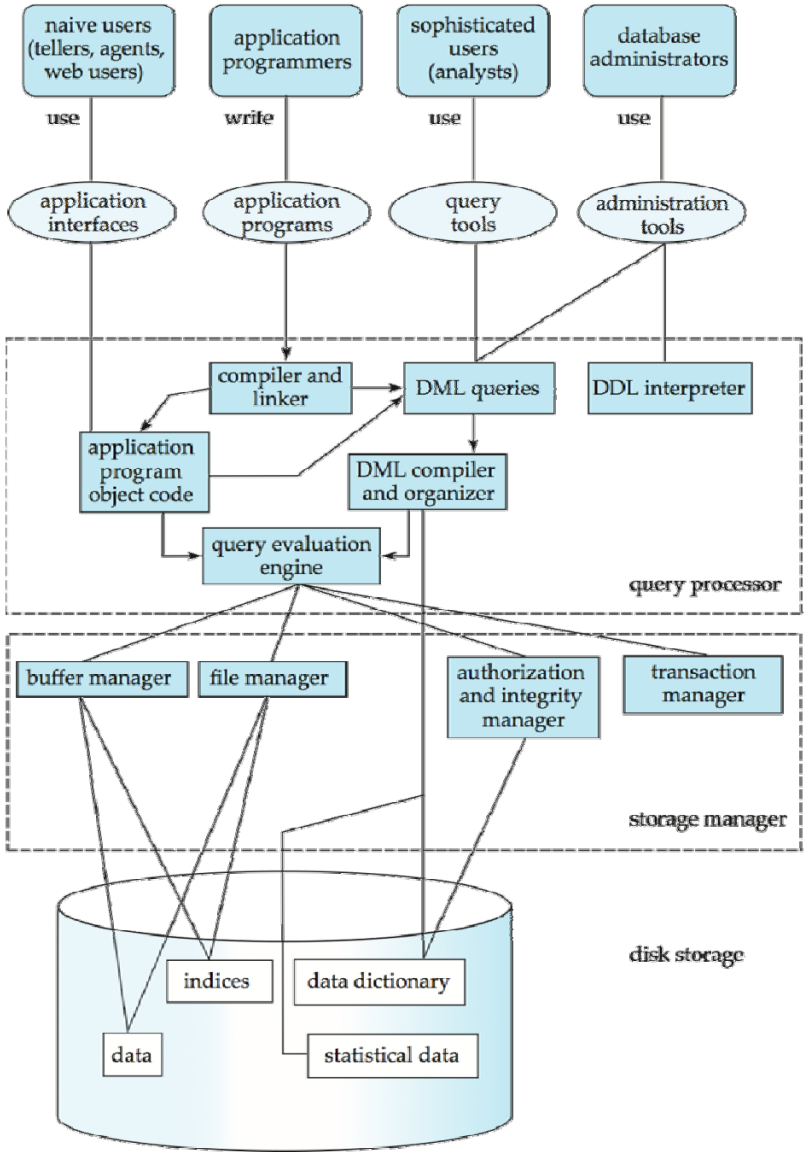
Transaction Management

- 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 Users and Administrators



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

Terminologies

- Data

A **symbolic record** that describes something

- DataBase（DB）

A **collection** of large amounts of data stored in a computer over a long period of time, organized and shareable

- Database Management System

A data management **software** between OS and user, include DDL, DML, database transaction and operation management, database create and maintain...

- Database System

A **system** consisting of databases, database management systems, applications, and database administrators that stores, manages, processes, and maintains data.

History of Database Systems

- 1950s and early 1960s:
  - Data processing using magnetic **tapes** for storage
    - Tapes provided only **sequential** access
  - P**unched 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 begins Ingres prototype
  - High-performance (for the era) transaction processing

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
- Early 2000s:
  - **XML** and XQuery standards
  - **Automated** database administration
- Later 2000s~:
  - Giant data storage systems BigData
    - Google BigTable, Yahoo PNuts, Amazon, .. NoSQL

End of Chapter 1