

# Database System Concept

# Chapter 1: Introduction

# Outline

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

# 引言

- Database Management System (DBMS)
- DBMS contains information about a particular enterprise
  - Collection of **interrelated** data(数据库)
  - **Set** of programs to access the data
  - 目的: An environment that is both *convenient* and *efficient* to use

- 设计数据库管理系统的目的：为了管理大量信息。
- 对数据的管理，涉及：
  - 信息**存储结构**的定义；
  - 信息**操作机制**的定义。

- Database systems are designed to **manage** large bodies of information.
- Management of data involves both
  - defining **structures** for storage of information
  - and providing mechanisms for the **manipulation** of information.
- the database system must ensure
  - the **safety** of the information stored,
  - despite system **crashes** or attempts at **unauthorized** access.
- If data are to be **shared** among several users, the system must **avoid** possible **anomalous** results.

# 1.1 数据库管理系统的应用

- 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
- 20世纪90年代，数据库改为web界面，提供在线服务和信息。
- 数据库的用户界面隐藏了访问数据库的细节
- Databases can be very large.
- Databases touch all aspects of our lives

# 1.2 数据库管理系统的目标

- In the early days, database applications were built directly on top of **file** systems
- University Database Example
- 所以，为了使用户可以对信息进行**操作**，系统应有一些对文件进行操作的**应用程序**；
- 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



- 典型的**文件处理系统**，是传统的操作系统所支持的。
- 永久**记录**被存储在多个不同的**文件**中，
- 人们编写不同的**应用程序**将记录从文件中**取出**或**加入**到适当的文件中。

# 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 different 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 10, 000) and updating it by withdrawing money (say 500, 100 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**

## 1.3 数据视图

- DBMS的一个主要目的就是给用户提供数据的**抽象视图**,
- 即, 系统**隐藏**关于数据**存储**和**维护**的某些细节。

## 1.3.1 数据抽象

- Levels of Abstraction用来屏蔽复杂性
- **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**;

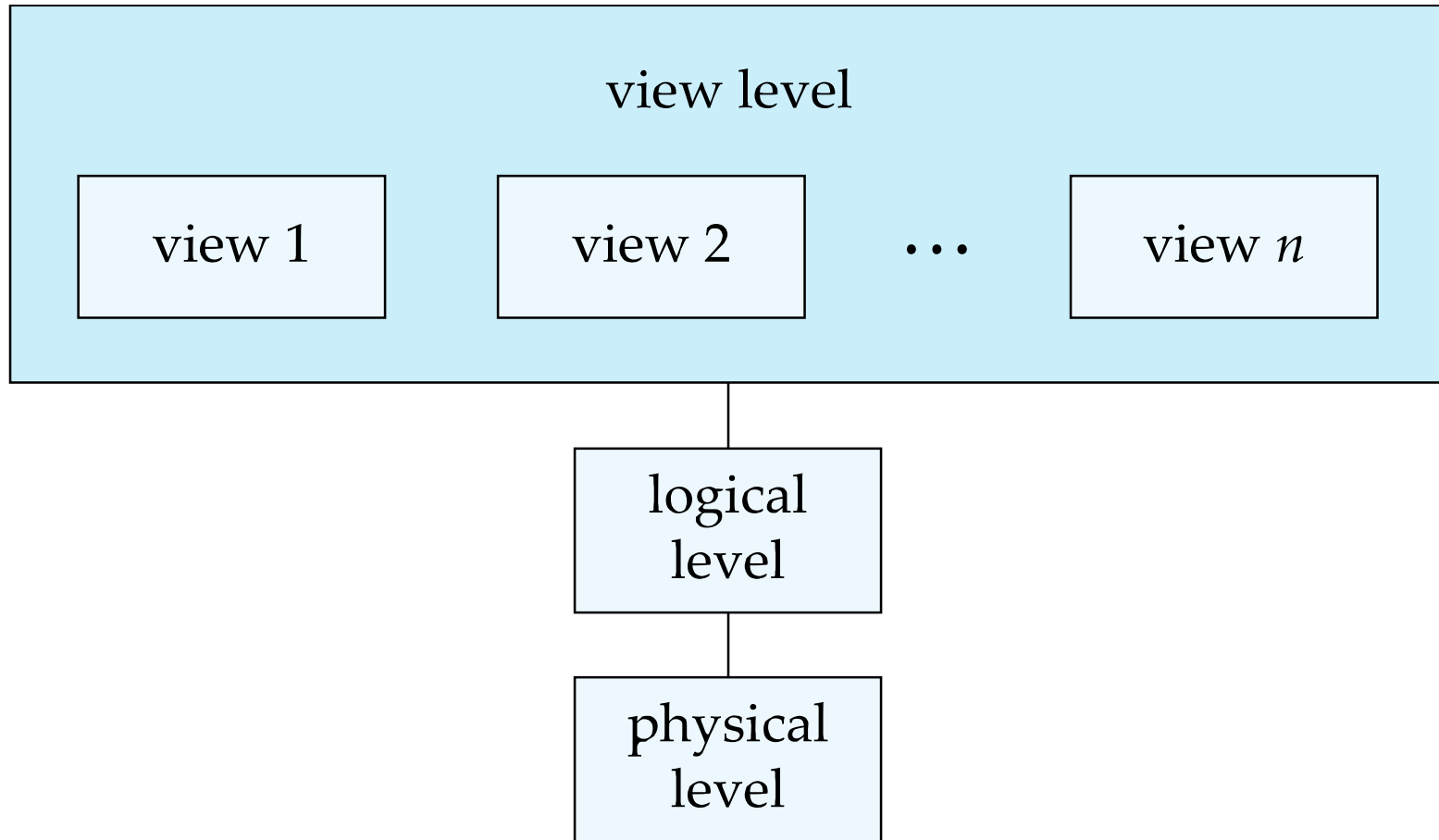
- 逻辑层就通过少量相对简单的结构**描述**了整个数据库。
- 虽然**逻辑层**的**简单**结构的实现可能涉及**复杂**的物理层结构，但逻辑层的**用户**不必知道这样的复杂性。
- 这称作**物理数据独立性** ( physical data independence)。

- **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





- At the **physical** level, a **record** can be described as a **block** of consecutive storage locations.
- The **compiler** hides this level of detail from programmers.
- the **database** system hides many of the lowest-level storage **details** from database programmers.
- Database **administrators**, may **be aware of** certain **details** of the physical organization of the data.

## 1.3.2 Instances and Schemas

- Similar to types and variables in programming languages
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable
- The overall design of the database is called the **database schema**
- **Physical schema**– the overall physical structure of the database

- **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
- schemas at the view level, sometimes called **subschemas**, that describe different views of the database.

- **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.

## 1.3.3 Data Models

- Underlying the structure of a database is the **data model**: A collection of conceptual tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Consistency constraints
- A data model provides a way to **describe** the design of a database at the physical, logical, and view levels.

- 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

# 关系模型( relational model)

- **关系模型**用表的集合来表示数据和数据间的联系。
- 每个表有多个**列**，每列有唯一的列名。
- 关系模型是基于**记录**的模型的一种。
- 每个表包含某种**特定类型**的记录。
- 每个记录类型定义了固定数目的**字段**（或属性）。
- 表的**列**对应于记录类型的属性。
- 关系数据模型是使用**最广泛**的数据模型。

# 实体—联系模型(entity-relationship model)

- 实体—联系(E-R)数据模型基于对现实世界的这样一种认识：
  - 现实世界由一组称作实体的基本对象以及这些对象间的联系构成。
  - 实体是现实世界中可区别于其他对象的一件“事情”或一个“物体”。
- 实体—联系模型被广泛用于数据库设计



# 基于对象的数据模型( object-based data model)

- 面向对象的数据模型可以看成是E-R模型增加了封装、方法（函数）和对象标识等概念后的扩展。
- The object-relational data model combines features of the object-oriented data model and relational data model

# 半结构化数据模型( senustructured data model)

- **半结构化**数据模型允许那些相同类型的数据项含有**不同的属性集**的数据定义。
- 可扩展标记语言( eXtensible Markup Language, XML)被广泛地用来表示半结构化数据。

# 1.4数据库语言

- 数据库系统提供
  - 数据定义语言（data-definition language）来**定义**数据库**模式**，
  - 以及数据操纵语言( data-manipulation language)来表达数据库的**查询和更新**。

## 1.4.1 Data Manipulation Language (DML)

- A **data-manipulation language (DML)** is a language that enables users to access and **manipulate (Read, Insert, Update, Delete)** the data organized by the appropriate data model
  - DML also known as query language

- 通常有两类基本的数据操纵语言：
  - 过程化DML( procedural DML), 要求用户指定需要什么数据以及如何获得这些数据。
  - 声明式DML( declarative DML) (也称为非过程化DML), 只要求用户指定需要什么数据, 而不指明如何获得这些数据。
- **查询( query)**是要求对信息进行检索的语句。
- DML中涉及**信息检索**的部分称作查询语言(query language)。

- **抽象层次**不仅可以用于**定义或构造**数据，而且还可以用于**操纵**数据。
- 在**物理层**，我们必须定义可**高效**访问数据的算法；
- 数据库系统的**查询处理器**部件将DML的查询语句翻译成数据库系统**物理层**的**动作序列**。
  -

## 1.4.2 数据定义语言

- 数据库**模式**是通过一系列**定义**来说明的，这些定义由一种称作**数据定义语言**(Data-Definition Language, DDL)的特殊语言来表达。
- 数据库系统所使用的**存储结构**和**访问方式**是通过一系列特殊的DDL语句来说明的，这种特殊的DDL称作**数据存储和定义**(data storage and definition)语言。
- 这些语句定义了数据库模式的**实现细节**，而这些细节对用户来说通常是**不可见的**。

- 存储在数据库中的数据值必须满足某些**一致性约束**（consistency constraint）。
- DDL语言提供了指定这种约束的**工具**。
- 每当数据库被**更新**时，数据库系统都会检查这些**约束**。
- **约束**可以是关于数据库的任意**谓词**。



- 数据库系统实现可以以**最小代价**测试的完整性约束，包括：
  - 域约束( domain constraint)。每个属性都必须对应于一个所有可能的**取值**构成的域。声明一种**属性**属于某种**具体的域**就相当于**约束**它可以取的值。
  - 参照完整性（referential integrity）。一个关系中**给定属性集**上的**取值**也在**另一关系**的某一属性集的取值中**出现**（参照完整性）。
    - 数据库的修改会导致参照完整性的**破坏**。
    - 当参照完整性约束被**违反**时，通常的处理是拒绝执行导致完整性被破坏的操作。

- **断言**( assertion)。一个**断言**就是数据库需要**时刻满足**的某一条件。
  - 域约束和参照完整性约束是断言的**特殊形式**。
  - 断言创建以后，系统会检测其有效性。
  - 如果**断言有效**，则以后只有**不破坏**断言的数据库**更新**才被允许。
- **授权**（authorization）。对于**不同的用户**在数据库中的**不同数据值**上允许**不同的访问类型**。
  - 读权限（read authorization）
  - 插入权限(insert authorization)
  - 更新权限(update authorization)
  - 删除权限(delete authorizahon)

- DDL以一些指令（语句）作为输入，生成一些输出。
- DDL的输出放在**数据字典**( data dictionary)中
- **数据字典**包含了元数据(metadata)
- **元数据**是关于数据的数据。
- 可把数据字典看作一种**特殊的表**，这种表只能由数据库系统本身（不是常规的用户）来**访问和修改**。
- 在读取和修改**实际**的数据前，数据库系统先要**参考**数据字典。

# 1.5 关系数据库

- 关系数据库基于**关系模型**，使用一系列**表**来表达**数据**以及这些数据之间的**联系**。
- 关系数据库也包括DML和DDL。

## 1.5.1 表

- 每个表有多个列，每个列有唯一的名字
- 每个表包含一种特定类型的记录。
- 每种记录类型定义固定数目的字段或属性。
  -
- 表的列对应记录类型的属性

# 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

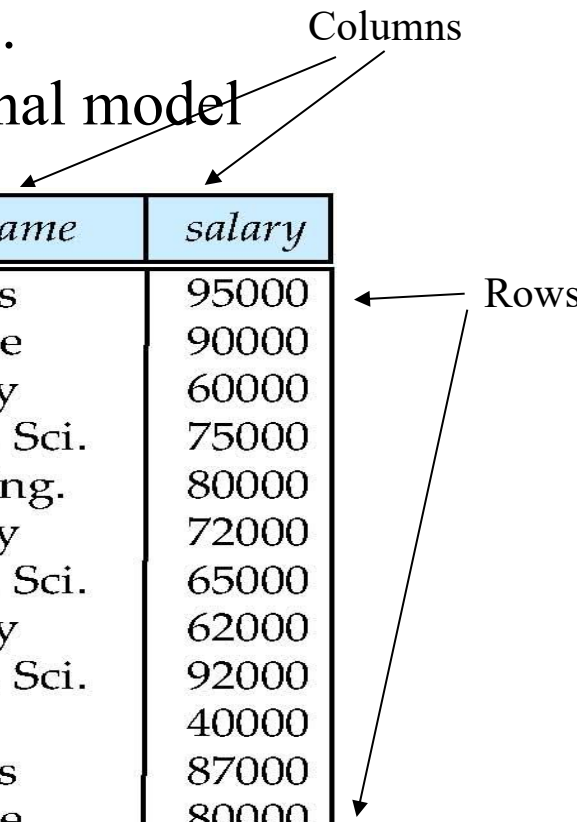
(a) The *instructor* table

<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

# Relational Model

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



<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

- 表可以如何存储在文件中。
  - 例如，
    - 一个特殊的字符（比如逗号）可以用来分隔记录的不同属性，
    - 另一特殊的字符（比如换行符）可以用来分隔记录。
- 对于数据库的开发者和用户，**关系模型**屏蔽了这些低层实现细节。



## 1.5.2 数据操纵语言

- SQL查询语言是非过程化的。
- SQL以几个表作为输入（也可能只有一个），总是仅返回一个表。

# 1.5.3 Data Definition Language (DDL)

- SQL提供了一个丰富的DDL语言，通过它，我们可以定义表、完整性约束、断言，等等
- 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

## 1.5.4 来自应用程序的数据库访问

- SQL is The most widely used commercial 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
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database and retrieve the results
  - Language extensions to allow embedded SQL, 通常用一个特殊字符作为DML调用的开始, 并且通过预处理器, 称为**DML预编译器** (DML precompiler), 来将DML语句转变成宿主语言中的**过程调用**。

# 1.6 Database Design

- 数据库设计的主要内容是**数据库模式**的设计。

## 1.6.1 设计过程

- 高层的**数据模型**为数据库**设计者**着提供了一个**概念框架**，其中
  - 说明了数据库用户的**数据需求**，
  - 以及将**怎样构造**数据库结构以满足这些需求。
- 因此，数据库设计的**初始阶段**是
  - 全面**刻画**预期的数据库用户的**数据需求**。
  - 为了完成这个任务，数据库设计者有必要和领域专家、数据库用户广泛地**交流**。
  - 这个阶段的成果是制定出用户需求的**规格文档**。

- 下一步，设计者选择一个**数据模型**，并运用该选定的数据模型的**概念**，将那些需求转换成一个数据库的**概念模式**。
  - 在这个**概念设计**（conceptual-design）阶段开发出来的**模式**提供了企业的**详细概述**。
  - 设计者再**复审**这个模式，确保所有的数据需求都**满足**并且相互之间**没有冲突**，
  - 在检查过程中设计者也可以去掉一些**冗余**的特性。
  - 这一阶段的**重点**是描述**数据**以及它们之间的**联系**，而**不是**指定物理的存储细节。

- 从**关系模型**的角度来看，**概念设计**阶段涉及
  - 决定数据库中应该包括哪些**属性**，
  - 以及如何将这些属性**组织**到多个表中。
  - 前者基本上是商业的决策，
  - 而后者主要是计算机科学的问题，解决这个问题主要有两种方法：
    - 一种是使用**实体—联系模型**
    - 另一种是引入一套算法（通称为**规范化**），这套算法将所有**属性集**作为输入，生成一组关系表。

- 一个开发完全的概念模式还将指出企业的功能需求。
- 在功能需求说明( specification of functional requirement)中，用户描述数据之上的各种操作（或事务），
  - 例如更新数据、检索特定的数据、删除数据等。
- 在概念设计的这个阶段，设计者可以对模式进行复审，确保它满足功能需求。

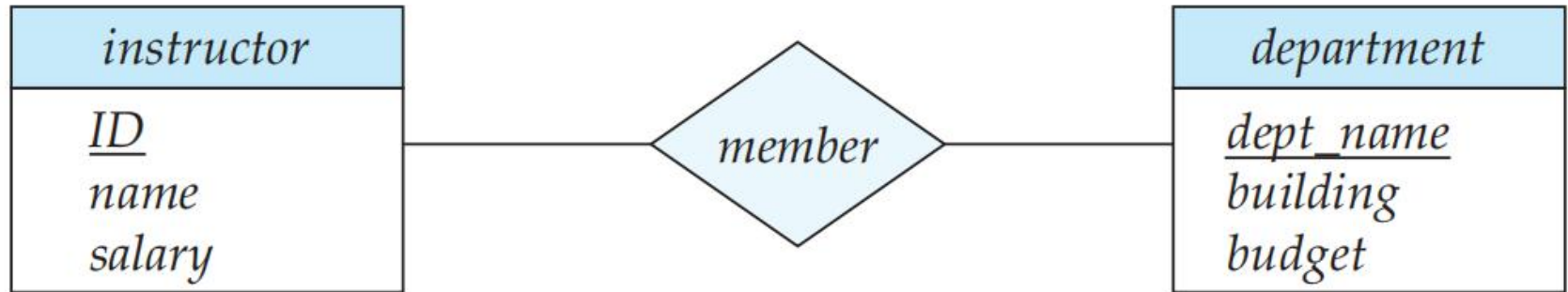


- 将抽象**数据模型**转换到**数据库实现**，进入最后**两个**设计阶段。
  - 在**逻辑设计阶段**（logical-design phrase），设计者将高层的**概念模式**映射到要使用的**数据库系统的实现数据模型**上；
  - 然后，设计者将得到的特定于系统的数据库模式用到**物理设计阶段**（physical-design phrase）中，在这个阶段中指定数据库的**物理特性**，
    - 包括**文件组织的形式**以及内部的**存储结构**，

## 1.6.3 实体—联系模型

- **实体—联系(E-R)数据模型**使用一组称作**实体**的基本对象，以及这些对象间的**联系**。
- 数据库中**实体**通过**属性(attribute)**集合来描述。
- **联系(relation)**是几个实体之间的**关联**。
- **同一类型**的所有**实体**的集合称作**实体集** (entity set) ，
- 同一类型的所有**联系**的集合称作**联系集** (relation set)。

- 数据库的总体**逻辑结构**（模式）可以用实体—联系图（entity- relationship diagram. E-R图）进行图形化表示。
- **统一建模语言**( Unified Modeling Language, UML)
  - **实体集**用**矩形框**表示，实体名在头部，属性名列在下面。
  - **联系集**用连接一对相关的实体集的**菱形**表示，联系名放在菱形内部。



**Figure 1.3** A sample E-R diagram.

- 除了实体和联系外，E-R模型还描绘了数据库必须遵守的对其内容的**某些约束**。
- 一个重要的约束是**映射基数** (mapping cardinality)，它表示通过某个联系集能与一实体进行关联的**实体数目**。

## 1.6.4 规范化

- 它的**目标**是生成一个**关系模式集合**，使我们存储信息时**没有**不必要的**冗余**，同时又能很轻易地**检索**数据。
- 这种方法是设计一种**符合**适当的**范式**（normal form）的**模式**，
  - 为确定一个**关系模式**是否**符合**想要的范式，我们需要**额外的**关于用数据库**建模**的现实世界中机构的信息。
  - 最常用的方法是使用**函数依赖**(functional dependency)

# Database Design (Cont.)

- Is there any problem with this relation?

<i>ID</i>	<i>name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
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

# Database Engine

- Storage manager
- Query processing
- Transaction manager

# 1.7 数据存储和查询

- 数据库系统划分为**不同的模块**，每个模块完成整个系统的一个功能。
- 数据库系统的功能部件大致可分为
  - **存储管理器**和**查询处理部件**
- 存储管理中，数据库系统对数据的组织必须满足使**磁盘和主存**之间数据的**移动最小化**。



# 1.7.1 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
  - Efficient **storing**, **retrieving** and **updating** of data

- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

- 存储管理部件包括：
  - 权限及完整性管理器(authorization and integrity manager)，它检测是否满足完整性约束，并检查试图访问数据的用户的权限。
  - 事务管理器( transaction manager)，它保证即使发生了故障，数据库也保持在一致的（正确的）状态，并保证并发事务的执行不发生冲突。
  - 文件管理器( ffile manager)，它管理磁盘存储空间的分配，管理用于表示磁盘上所存储信息的数据结构。

- **缓冲区**管理器(buffer manager)，它负责将数据从磁盘上**取到**内存中来，并决定哪些数据应被缓冲存储在内存中。

- 存储管理器实现了几种数据结构，作为系统物理实现的一部分：
  - 数据文件( data files)，存储数据库自身。
  - 数据字典( data dictionary)，存储关于数据库结构的元数据，尤其是数据库模式。
  - 索引( index)，提供对数据项的快速访问。数据库索引提供了指向包含特定值的数据的指针。

## 1.7.2 查询处理器

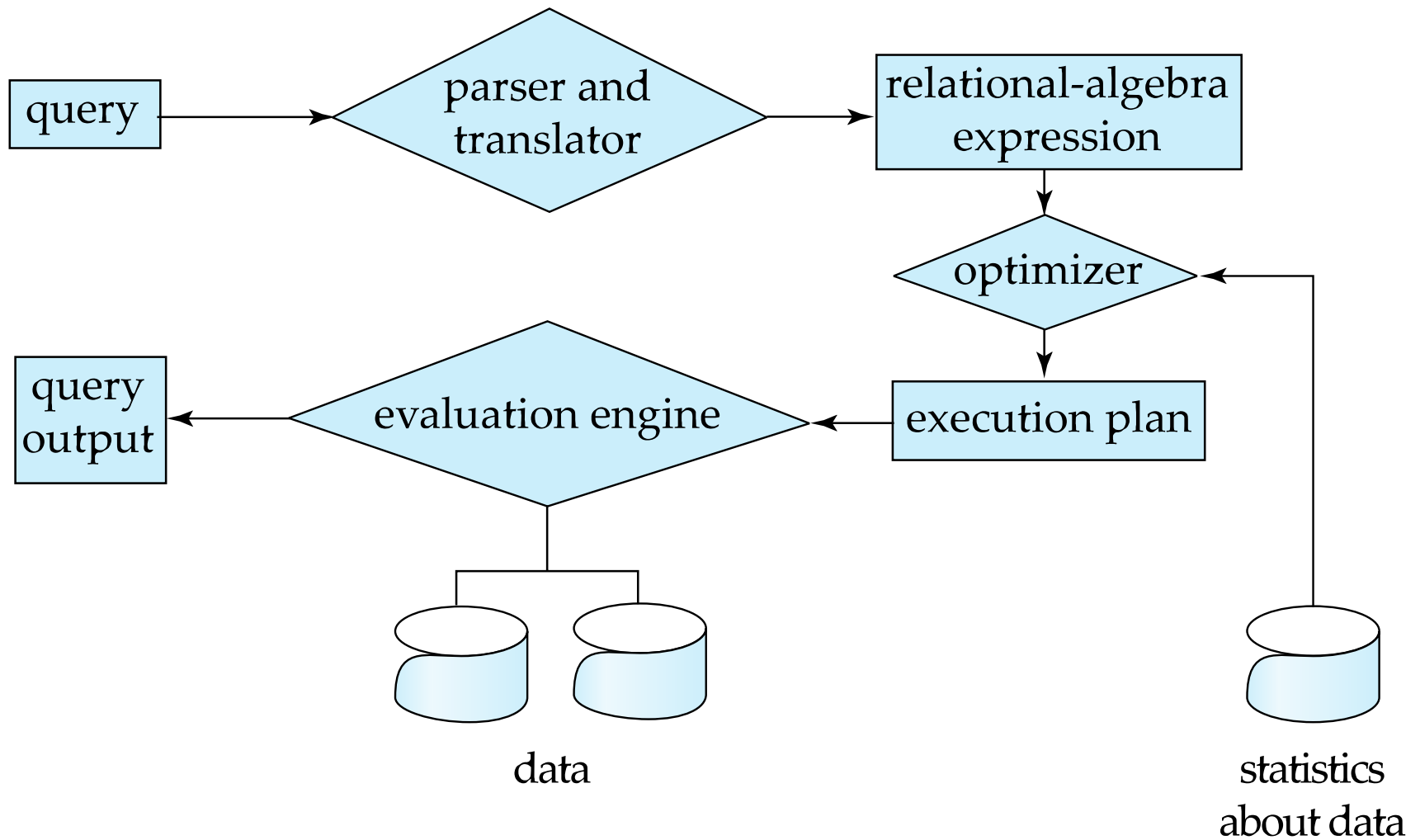
- 查询处理器组件包括：
  - DDL解释器(DDL interpreter). 它解释DDL语句并将这些定义记录在数据字典中。
  - DML编译器(DML compiler), 将查询语言中的DML语句翻译为一个执行方案, 包括一系列查询执行引擎能理解的低级指令。
    - 一个查询通常可被翻译成多种等价的具有相同结果的执行方案的一种。
    - DML编译器还进行查询优化 (query optimization), 也就是从几种选择中选出代价最小的一种。

- 查询执行引擎（query evaluation engine），执行由DML编译器产生的低级指令。

# 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

# 1.8 Transaction Management

- What if the system fails?
- What if more than one user is concurrently updating the same data?
- 通常，对数据库的几个操作**合起来**形成一个逻辑单元。
- 这种要么完成要么不发生的要求称为**原子性**（atomicity）。
- 这种正确性的要求称作**一致性**（consistency）。
- 当操作成功结束后，即使发生系统**故障**，数据也应该保持操作成功结束后的**新值**。这种**保持**的要求称作**持久性**（durability）。

- A **transaction** is a collection of operations that performs a single logical function in a database application.
- 每一个事务是一个既具原子性又具一致性的单元。
- 因此，要求事务不违反任何的数据库一致性约束，
  - 即，如果事务启动时数据库是一致的，那么当这个事务成功结束时数据库也应该是一致的。

- 定义各个事务是程序员的职责，事务的定义应使之能保持数据库的一致性。
- 原子性和持久性的保证是数据库系统的职责，确切地说，是恢复管理器(recover manager)的职责。
- 故障发生时，失败的事务必须对数据库状态不产生任何影响。
- 因此，数据库必须被恢复到该失败事务开始执行以前的状态。

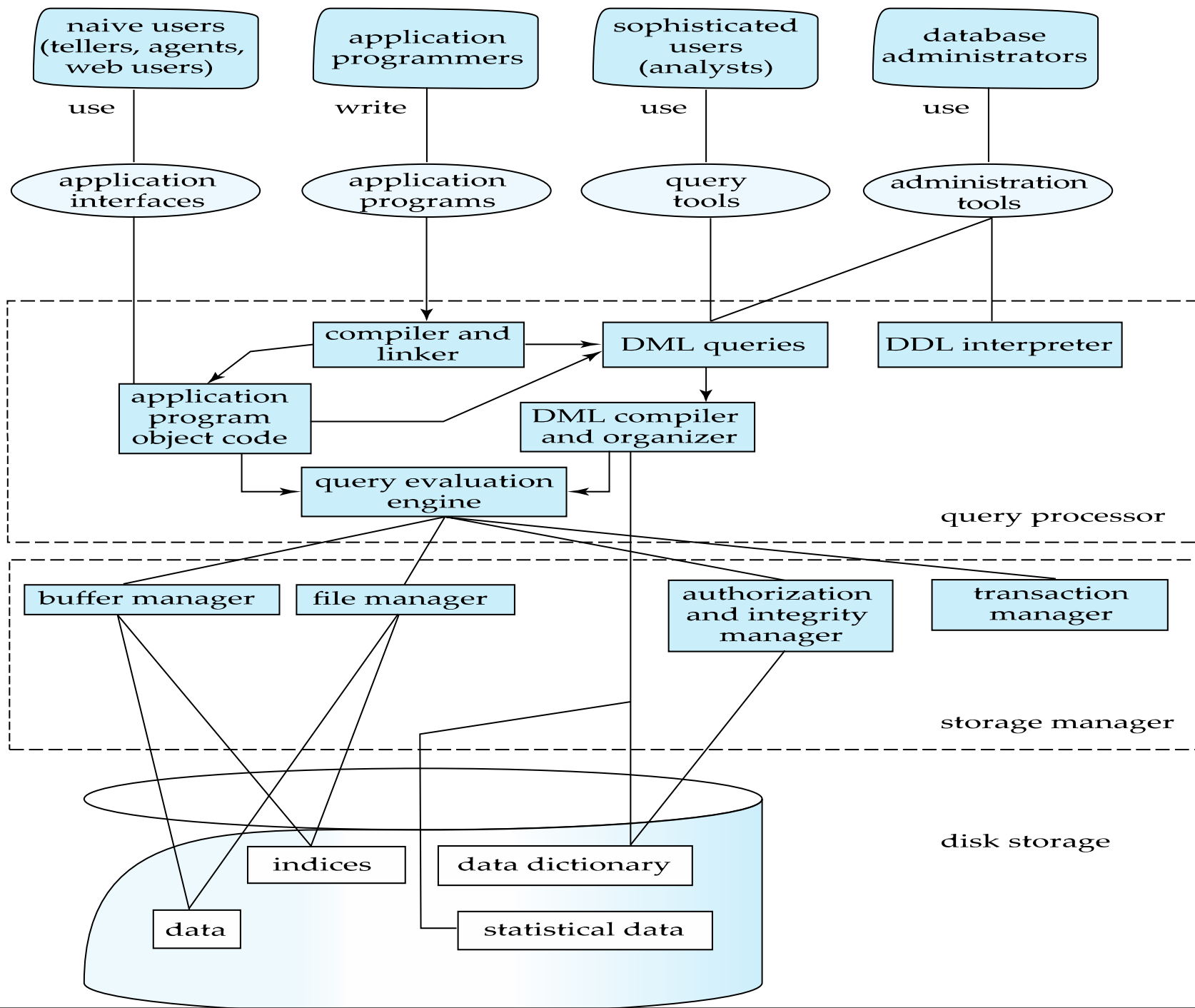
- 数据库系统必须进行故障恢复( failure recovery), 即检测系统故障并将数据库恢复到故障发生以前的状态。
- **Transaction-management component** ensures that the database remains in a consistent (correct) state, despite system failures (e.g., power failures and operating system crashes) and transaction failures.

- 当多个事务 **同时** 对数据库进行更新时，即使每个 **单独的** 事务都是 **正确** 的，数据的一致性也可能被 **破坏**。
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.

# 1.9 数据库体系结构

- 数据库系统的体系结构很大程度上取决于数据库系统所运行的计算机系统。
- 数据库系统可以是
  - 集中式的
  - 客户 / 服务器式的（一台服务器为多个客户机执行任务）
  - 也可以针对并行计算机体系结构设计数据库系统；
  - 分布式数据库包含地理上分离的多台计算机。

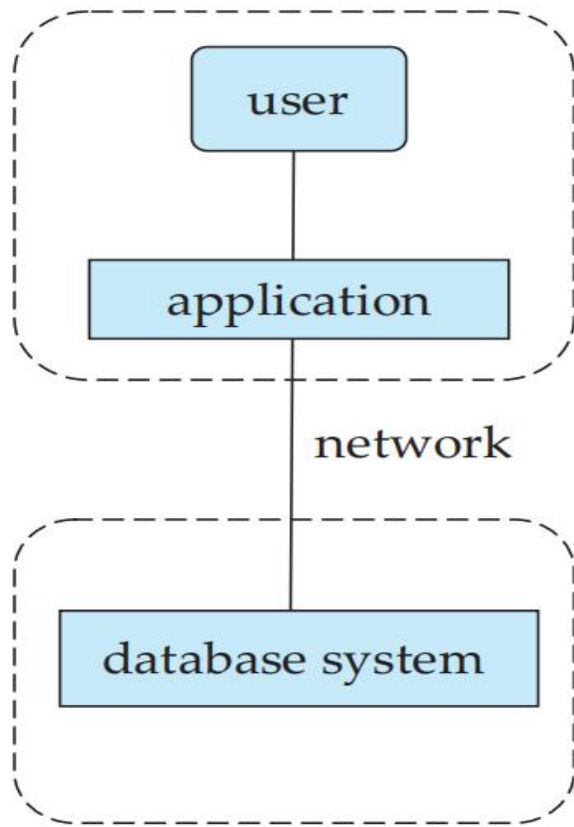




## 1.9.1 客户/服务器系统

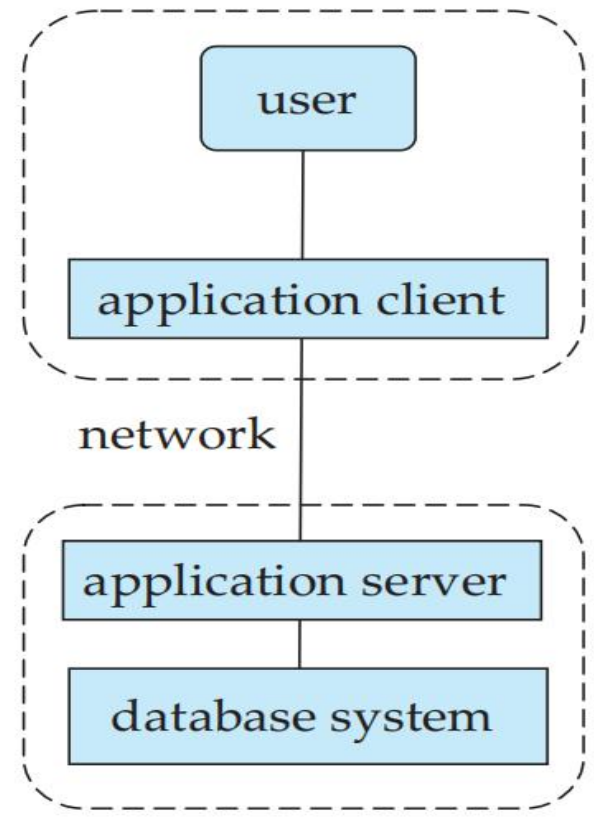
- 数据库系统的大多数用户是通过网络与数据库相连。
- 因此，可区分
  - 远程数据库用户工作用的客户机( client)
  - 和运行数据库系统的服务器(server)。

- 数据库应用通常可分为两或三个部分



(a) Two-tier architecture

client



(b) Three-tier architecture

server

**Figure 1.6** Two-tier and three-tier architectures.

- 在一个两层体系结构（two-tier architecture）中，应用程序驻留在客户机上，通过查询语言表达式来调用服务器上的数据库系统功能。
- 像ODBC和JDBC这样的应用程序接口标准被用于进行客户端和服务器的交互。

- 在一个三层体系结构( three- tier architecture)中，
- 客户机只作为一个前端并且不包含任何直接的数据库调用。
- 客户端通常通过一个表单界面与应用服务器(application server)进行通信。
- 而应用服务器与数据库系统通信以访问数据。

- 应用程序的**业务逻辑**（business logic），也就是说在**何种条件**下做出**何种反应**，被嵌入到**应用服务器**中，**而不是**分布在多个**客户机**上。
- **三层结构**的应用更适合**大型应用**和**互联网**上的应用。

## 1.9.2 并行数据库系统

- **并行系统**通过并行地使用多个处理器和磁盘来提高速度和I/O速度。
- 并行数据库系统，用来查询大型数据库（TB），高速处理大量事务（千个事务/秒）
- 并行处理中，许多操作是**同时执行**的。
- **粗粒度**（coarse-grain），并行机由少量**强大**处理器组成；
- **大规模并行**（massive parallel）机，或称**细粒度并行**（fine-grain parallel）机，使用数千小型处理器。
  -

- 对数据库系统性能的度量有两种方式：
  - 吞吐量（throughput）：在给定时间内所能完成任务的数量；
  - 响应时间（response time）：单个任务从提交到完成所需的时间。



## 1.9.3 分布式数据库系统

- **分布式数据库系统**（distributed database system），数据库存储在**多台**计算机中，通过**高速私有网络**或**互联网**连接（通讯），它们**不共享**主存储器和磁盘。
- 建立分布式数据库系统的**原因**：
  - **数据共享**（sharing Data），使得一个站点（site）上的用户可以访问其他站点的数据；
  - **自治性**（autonomy），每个站点对本地存储的数据保持一定程度的控制。
    - 分：全局数据库管理员和本地数据库管理员

- 可用性（availability），如果一个站点发生故障，其他站点还可以运行。
  - 如果数据项在几个站点上进行了复制（replicate），那么，该数据可以在这些站点中的任何一个上找到，一个故障，整个系统还可以运行。

# 1.10 数据挖掘与信息检索

- **数据挖掘**(data mining), 指**半自动地**分析大型**数据库**并从中找出**有用的模式**的过程。
  - 和**知识发现**(也称为**机器学习**( machine learning))或者**统计分析**一样, 数据挖掘试图从数据中**寻找规则或模式**。
- 在数据挖掘中还需要**人参与**, 包括
  - **数据预处理**使数据变为适合算法的格式,
  - 在已发现模式的**后处理**中找到新奇的**有用模式**。
  - 给定一个数据库, 可能有**不止一种**类型的模式, 需要人工交互**挑选**有用类型的模式。

- 商业上，已经开始利用联机数据来支持对于业务活动的更好的决策。
- 大型企业有各种不同的可用于业务决策的数据来源。
- 要在这些各种各样的数据上高效地执行查询，企业建立了数据仓库( data warehouse)。
  -
- 数据仓库从多个来源收集数据，建立统一的模式，驻留在单个节点上。
- 这些数据中就包含文本数据

- 文本数据是**非结构化的**，与关系数据库中严格的结构化数据不同。
- **查询**非结构化的**文本数据**被称为**信息检索**（information retrieval）。
- **信息检索系统**和**数据库系统**很大程度上是**相同**的——特别是基于**辅助存储器**的数据存储和检索。
- 但是信息系统领域与数据库系统所**强调**的**重点**是不同的，
  - **信息系统**重点强调
    - 基于**关键词**的**查询**，
    - 文档与查询的**相似度**，
    - 以及文档的**分析、分类和索引**。

# 1.11 特种数据库

## 1.11.1 基于对象的数据模型

- **面向对象数据模型**( object- based data model)可以看作E-R模型的**扩展**，增加了
  - 封装、方法（函数）和对象标识。
  - 继承、对象标识和信息封装（信息隐蔽），
  - 以及对外提供方法作为访问对象的接口

## 1.11.2 半结构化数据模型

- **半结构化数据模型**，允许那些**相同**类型的数据项有**不同的**属性集的数据说明。
- XML提供了表达含有**嵌套结构**的数据的方法，能够**灵活**组织数据结构，这对于一些**非传统数据**来说非常重要。

# 1.12 数据库用户和管理员

- 数据库系统的一个主要目标是
  - 从数据库中检索信息和往数据库中存储新信息。
- 使用数据库的人员可分为
  - 数据库用户
  - 和数据库管理员。



# 1.12.1 数据库用户和用户界面

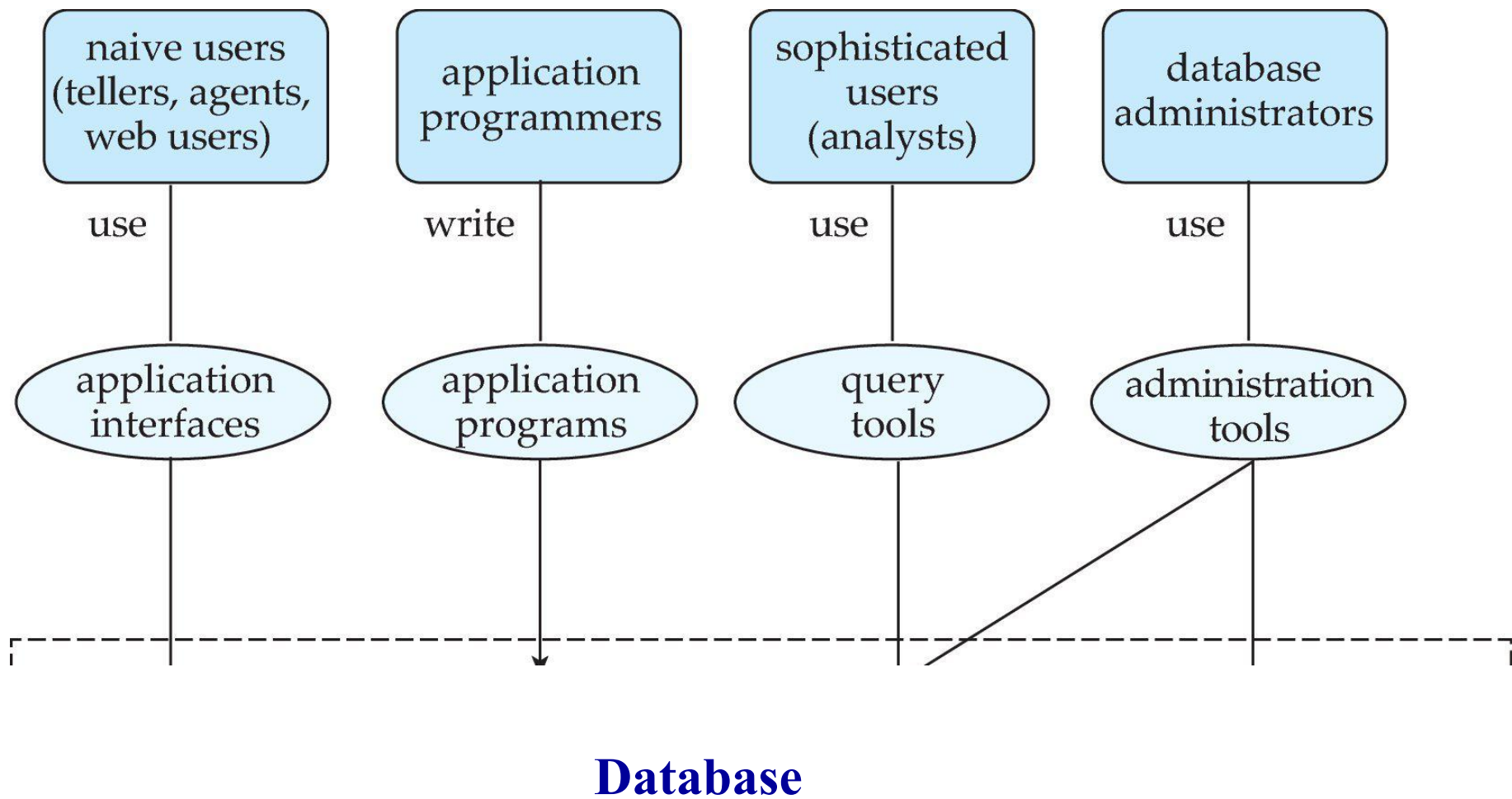
- 根据所期望的与系统交互方式的不同，数据库系统的用户可以分为四种不同类型。
- 系统为不同类型的用户设计了不同类型的用户界面。
  - 无经验的用户( naive user)是默认经验的用户，他们通过激活事先已经写好的应用程序同系统进行交互
    - 此类用户的典型用户界面是表格界面，用户只需填写表格的相应项就可以了

- 应用程序员(application programmer)是编写应用程序的计算机专业人员。
  - 快速应用开发(Rapid Application Development, RAD)工具使应用程序员能够尽量少编写程序就可以构造出表格和报表。
- 老练的用户(sophisticated user), 不通过编写程序来同系统交互, 而是用数据库查询语言或数据分析软件这样的工具来表达他们的要求。
  - 分析员通过提交查询来研究数据库中的数据, 所以属于这一类用户。

– 专门的用户( specialized user)是老练的用户，他能够编写专门的、不适合于传统数据处理框架的数据库应用。

- 这样的应用包括：

- 计算机辅助设计系统
- 知识库和专家系统
- 存储复杂结构数据（如图形数据和声音数据）的系统，
- 以及环境建模系统。



## 1.12.2 数据库管理员

- 使用DBMS的一个主要原因是可以对数据和访问这些数据的程序进行集中控制。
- 对系统进行集中控制的人称作数据库管理员（DataBase Administrator, DBA）。
- DBA的作用包括：
  - 模式定义(schema definition)。DBA通过用DDL书写的一系列定义来创建最初的数据库模式。
  - 存储结构及存取方法定义( storage structure and access-method defmition)。

- 模式及物理组织的修改（schema and physical-organization modification）。
- 数据访问授权（granting of authorization for data access）
  - - 通过授予不同类型的权限，DBA可以规定不同的用户各自可以访问的数据库的部分。
    - 授权信息保存在一个特殊的系统结构中，一旦系统中有访问数据的要求，数据库系统就去查阅这些信息。

## —日常维护( routine maintenance)。

» 数据库管理员的**日常维护**活动有：

- **定期备份**数据库，或者在磁带上或者在远程服务器上，以防止像洪水之类的灾难发生时数据丢失。
- 确保正常运转时所需的**空余磁盘空间**，并且在需要时**升级**磁盘空间。
- **监视**数据库的**运行**，并确保数据库的**性能**不因一些用户提交了花费时间较多的任务就**下降**很多。

End of Chapter 1