



# **DATABASE MANAGEMENT SYSTEM CO204**

**B. Tech. II CSE (Sem-4<sup>th</sup>)  
COED**

**Credits: 5 (L:3 T:1 P:2)**



# Goals

- Learn about the foundations of Relational DBMS
  - Design methodology for databases and their structural correctness
  - Applying the theory behind various database models
  - Declarative programming: specify WHAT you want, not HOW to get it
  - Data independence
  - Recovery from crashes to a consistent state
  - Programming for concurrent execution: transactions
- Be able to create, access, and manipulate a database through SQL, PL/SQL and from an application
- Be able to understand and critically evaluate features of competing data management offerings
- Working in group settings to design and implement database projects



# Outcome of the Course

1. Fluency in writing code in **SQL** for data definition, queries, updates.
2. Able to design the **entity-relationship model** for database design.
3. Able to apply the principles of **normalization** to relational database design.
4. Able to use the results of **database optimizer plans** analyze and tune database system performance by appropriate selection of indices.
5. Understanding of the concept of a database **transaction**, including atomicity, **concurrency, and recovery**.
6. Able to select appropriate **transaction isolation levels**.
7. Understanding of **SQL authorization** and its relationship to broader issues in database security.
8. Able to use **JDBC** to embed SQL database accesses in Java/web application code.



# Daily life Applications uses DBMS

- Daily life uses 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 touch all aspects of our lives



# University Database Example

- Application program examples, for the University
  - 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



# Database Management System (DBMS)

- Databases can be very large
- To Manage the data
  - 4 Store data
  - 4 Update data
  - 4 Answer questions about the data
- 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



# DBMS

- A DBMS is a system of programs

Includes facilities to:

1. Define and modify the database structure
2. Construct the database on a storage medium
3. Manipulate the database: queries and updates
4. Maintain integrity and security over the database by
  - Protecting data from inconsistency due to multiple concurrent users
  - Crash recovery
  - Access control

Done using log file / user roles



# Data Abstraction

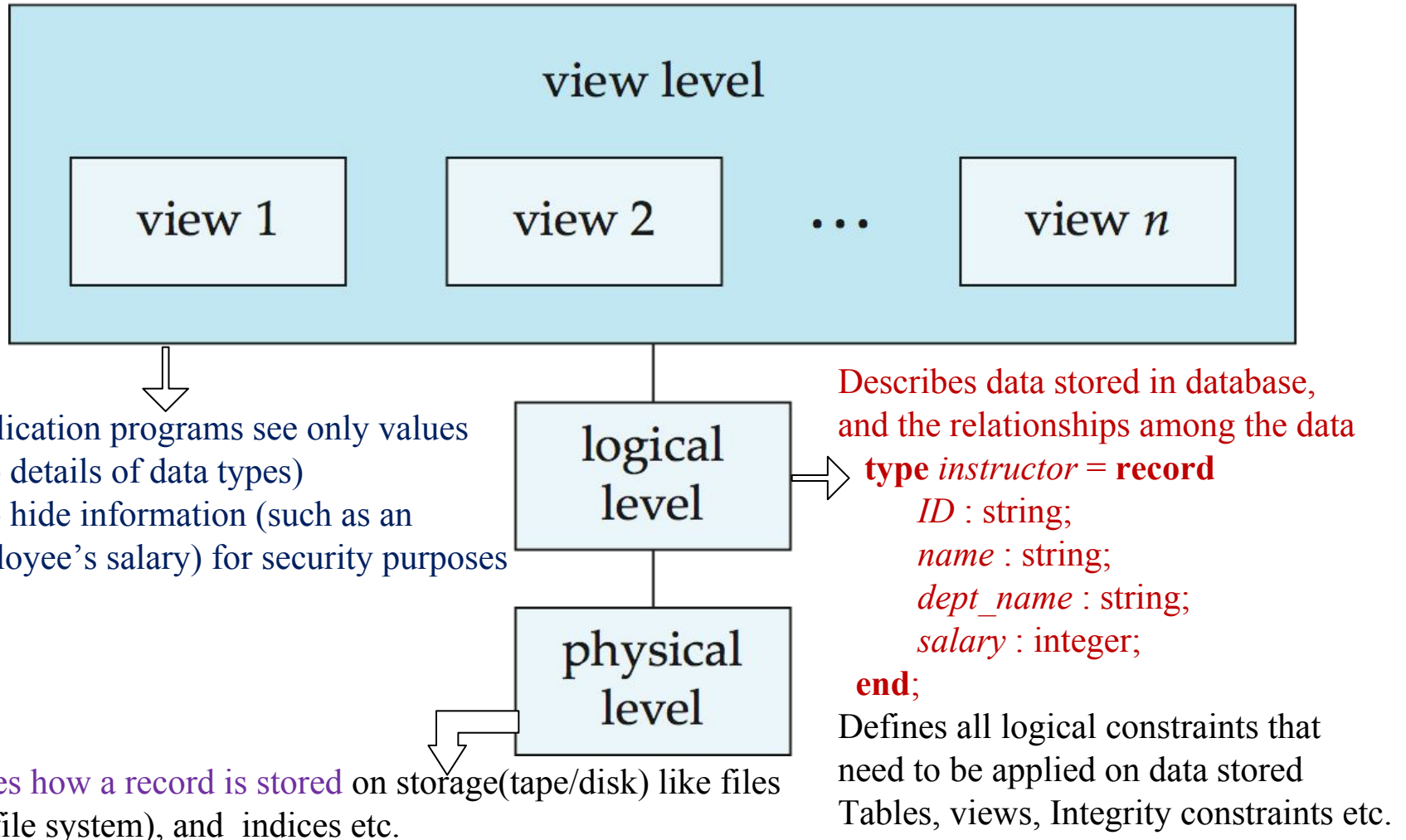
- Probably the most important purpose of a DBMS
- Hides low-level details from the users of the system





# View of Data

A High Level Data View for a database system





# Data Abstraction: Banking Example

- Logical level:
  - Provide an abstraction of tables
  - Two tables can be accessed:
    - 4 *accounts*
      - Columns: account number, balance
    - 4 *customers*
      - Columns: name, address, account number
- View level:
  - A teller (non-manager) can only see a part of the *accounts* table
    - 4 Not containing high balance accounts



# Data Abstraction: Banking Example

- Physical Level:
  - Each table is stored in a separate ASCII file
  - # separated fields
- Identical to what we had before ?
  - BUT the users are not aware of this
    - 4 They only see the tables
    - 4 The application programs are written over the tables abstraction
  - Can change the physical level without affecting users
  - In fact, can even change the logical level without affecting the *teller*



# Instances and Schemas

- Similar to types and variables in programming languages
- **Schema**
  - **Analogous to type information of a variable in a program**
  - **The logical structure of the database**
  - **Designed when database doesn't exist at all**
  - **Very hard to do any changes once the database is operational**
  - **Does not contain any data or information**
  - Example: The database consists of information about a set of customers and accounts and the relationship between them
  - **Physical schema:**
    - 4 Database design at the physical level
  - **Logical schema:**
    - 4 Database design at the logical level



# Instances and Schemas

- **Instance**
  - **Analogous to the value of a variable**
  - The actual content (or a state) of the database with data at a particular point in time
  - Snapshot of database
  - Tend to change with time
  - DBMS ensures that its every instance (state) must be a valid state
    - 4 By keeping up to all validation, constraints and condition that database designers has imposed or it is expected from DBMS itself

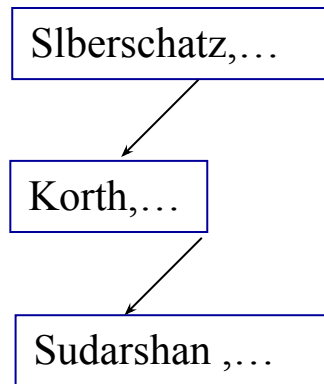


# Data Independence

- One of the most important benefits of using a DBMS
- Applications insulated from how data is structured and stored
- 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
  - e.g. The capability to change the lower level structure of data without having to change the application programs at the next higher level



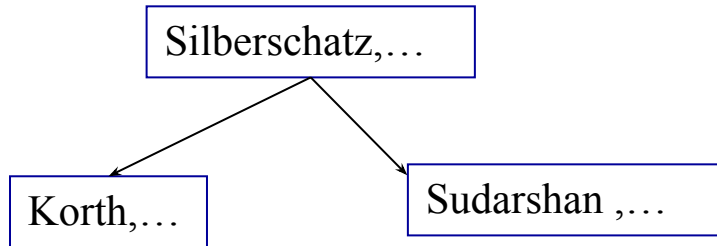
# Data Dependence



```
Search ( name-asked : char[30] );  
    s : Student-Record;  
    ...  
Begin  
    open ( student-file );  
    while not end of file do  
        begin  
            read-a-record ( s );  
            if s.name = name-asked then  
                begin  
                    print ( s );  
                    stop;  
                end;  
            end;  
        end;  
    print ( "No such a student." );  
End.
```



# Data Dependence



```
Search ( name-asked : char[30] );  
    root-record : student-record;  
begin  
    root-record := root record;  
    Find-It (name-asked, root-record );  
end.
```

```
Find-It ( name-asked : char[30];  
          root : student-record );  
begin  
    if root = null then  
        begin  
            print ( "No such a student." );  
            stop;  
        end;  
    if root.name = name-asked then  
        begin  
            print( root );  
            stop;  
        end  
    else if root.name < name-asked then  
        find-it ( name-asked,  
                  root.left );  
    else  
        find-it ( name-asked, root.right );  
    end.
```





# Data Independence

- Probably the most important purpose of a DBMS
- Hides low-level details from the users of the system
- **Physical Data Independence**
  - If, modify the physical schema
    - 4 No need to change the logical schema
  - Applications depend on the logical schema
  - Query and update logical structure, not physical structure
- **Logical data independence**
  - Protection from changes in logical structure of data
  - If logical structure changes
    - 4 Then, create view with old structure
  - Works fine for queries, but might be tricky for updates



# Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Types
  - 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:
    - 4 Network model
    - 4 Hierarchical model



# Relational Model

- Relational model
- 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

(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



# 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
  - **Procedural** – user specifies what data is required and how to get those data
  - **Declarative (nonprocedural)** – user specifies what data is required without specifying how to get those data
    - 4 SQL is the most widely used query language



# 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
    - 4 Primary key (ID uniquely identifies instructors)
    - 4 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



# Database Design

The process of designing the general structure of the database:

- Logical Design
  - Deciding on the database schema
  - 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?

- Is there any problem with this design?

<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



# Design Approaches

- Normalization Theory
  - Formalize what designs are bad, and test for them
- Entity Relationship Model
  - Entity
  - Relation
  - Models an enterprise as a collection of *entities* and *relationships*
  - Represents diagrammatically by an *entity-relationship diagram*



# ER Model

- Entity
  - A “thing” or “object” in the enterprise that is distinguishable from other objects
    - Described by a set of *attributes*
    - Examples: Korth, CO402
    - Form entity sets with other entities of the same type that share the same properties
    - Set of all people, set of all classes



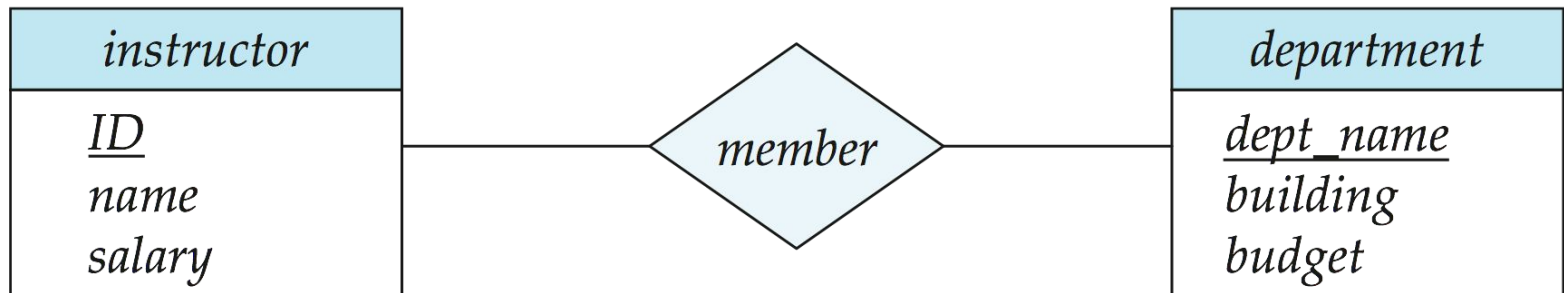
# ER Model

- Relation
  - Relate 2 or more entities
    - 4 E.g. Korth book used at SVNIT Computer Engg. Branch
  - Form relationship sets with other relationships of the same type that share the same properties
    - 4 Customers have accounts at Branches
  - Can have attributes:
    - 4 Customer has multiple accounts may have an attribute start-date
  - Can involve more than 2 entities
    - 4 Employee works at Branch as Manager



# The Entity-Relationship Model

- Example Partial Diagram:





# 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



# Storage Management

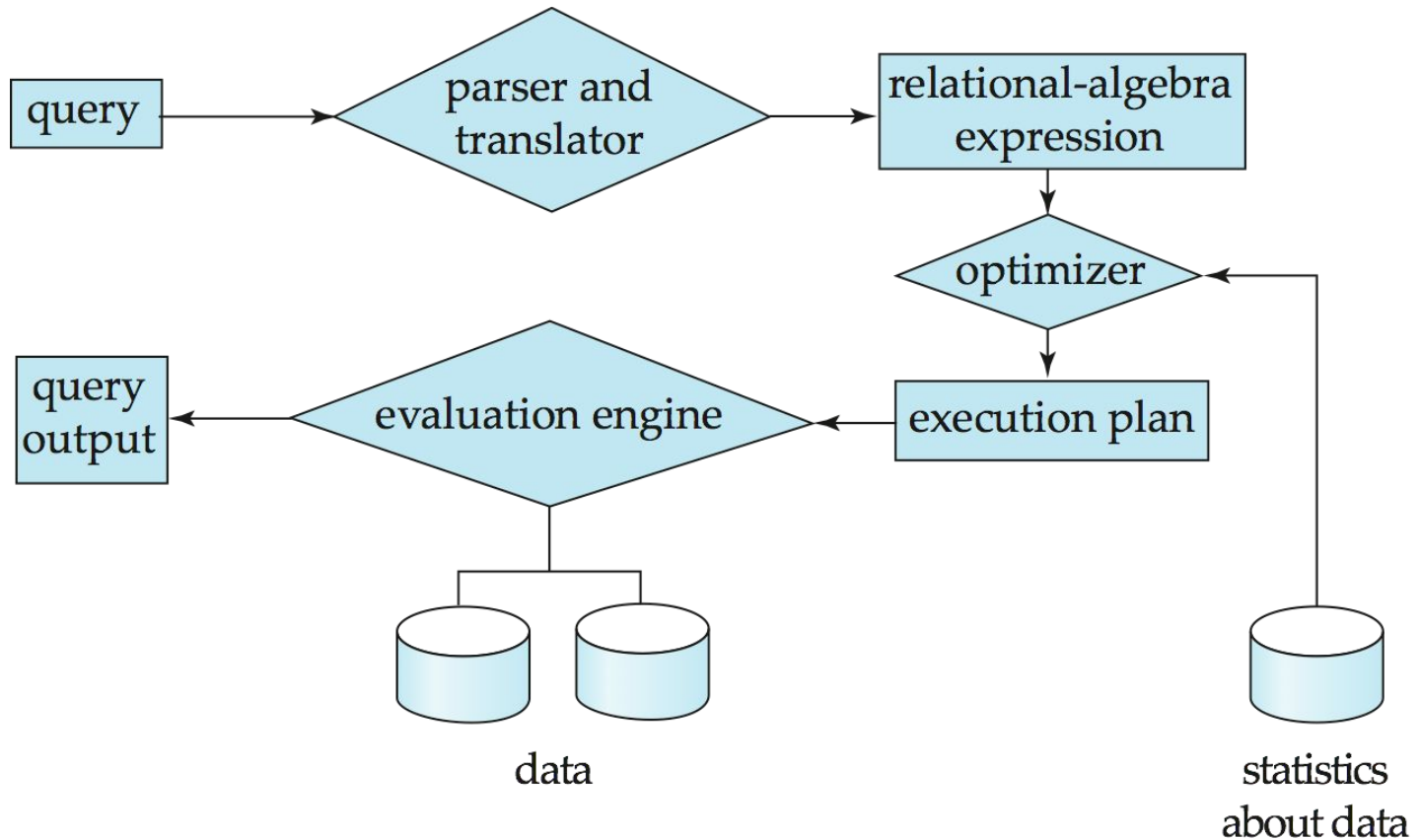
- **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system
- The storage manager is responsible to the following tasks:
  - Interaction with the file manager
  - Efficient storing, retrieving and updating of data
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing





# Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation





# 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**

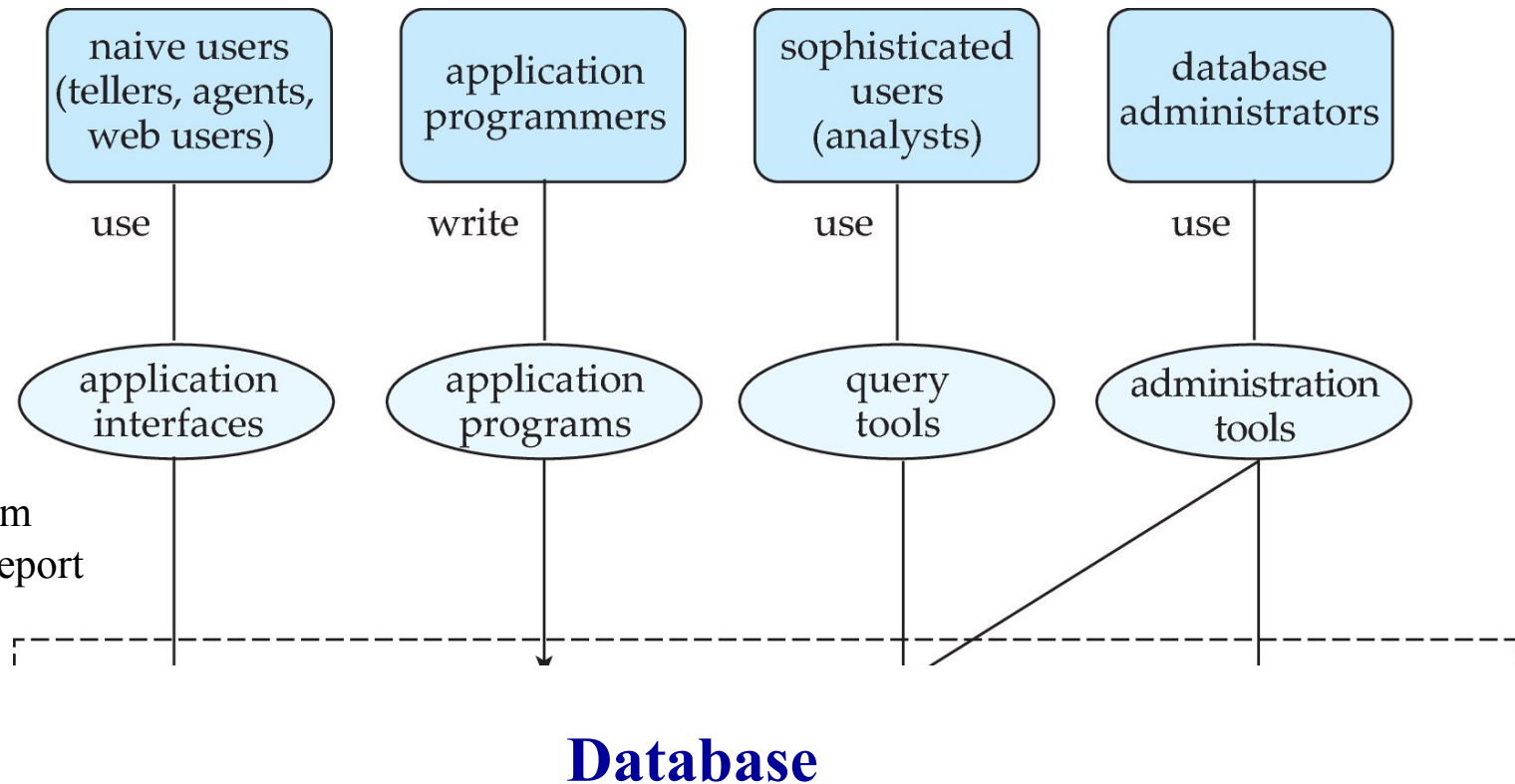


# Transaction Management

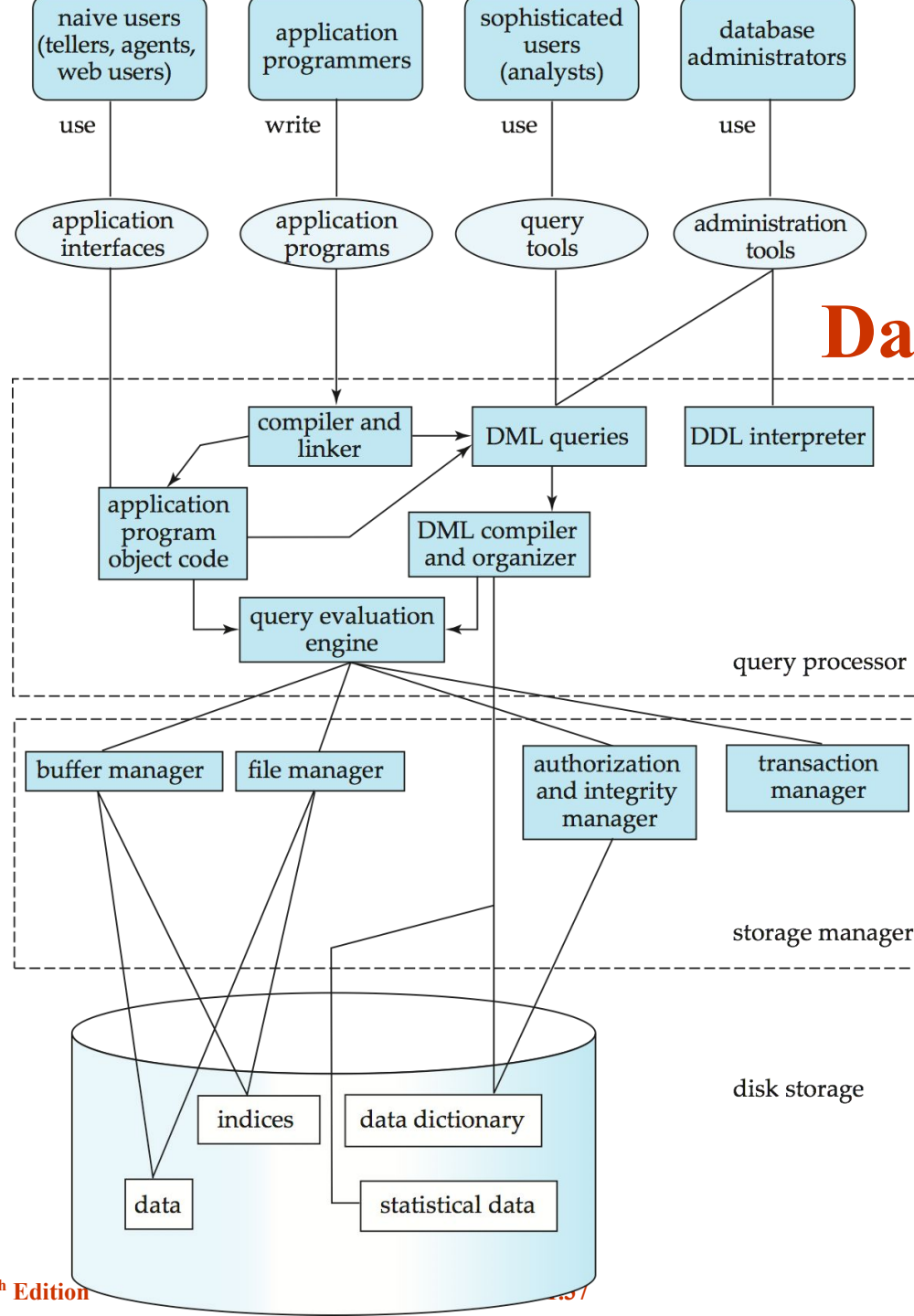
- What if the system fails?
- What if more than one user is concurrently updating the same data?
- **Transaction**
  - A collection of operations that performs a single logical function in a database application
- **Transaction-management component**
  - Ensures that the database remains in a consistent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



Tutorial Question: Enlist and distinguish the different database users in terms of Capability of data access, Tool used, etc.



# Database System Internals



# Database Administrator - Duties

- **Schema Definition**
  - Creates the original database schema using DDL statements
- **Storage structure and access method definition**
- **Schema and physical organization modification**
- **Granting of authorization** for data access
  - Which parts of the database various users can access
  - This information is consulted for granting information whenever someone attempts to access data in the system
- **Routine maintenance**
  - **Periodically backup** of database onto tapes, remote servers
    - To prevent data loss in case of disasters such as flooding
  - Ensures **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 Architecture

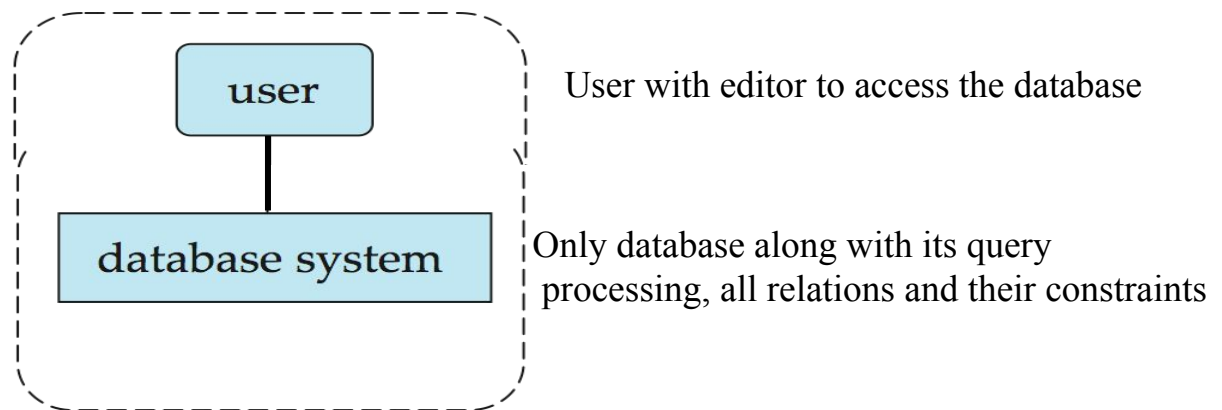
The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

- Centralized
  - All the DBMS functionality, application program execution and user interface processing of dumb terminals were carried out on a single centralized machine like mainframe machine
- Client-server
  - A client is typically a user machine not dumb terminal that provides user interface capabilities and local processing
  - A server is a system containing both hardware and software that can provide services to the client machines
- Parallel (multi-processor)
- Distributed



# Client Server Architecture

- Single tier or Multi tier
- n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed or replaced
- 1-tier architecture
  - DBMS is the only entity where user directly sits on DBMS and uses it
  - Any changes done here will directly be done on DBMS itself
  - Does not provide handy tools for end users and preferably database designer and programmers use single tier architecture

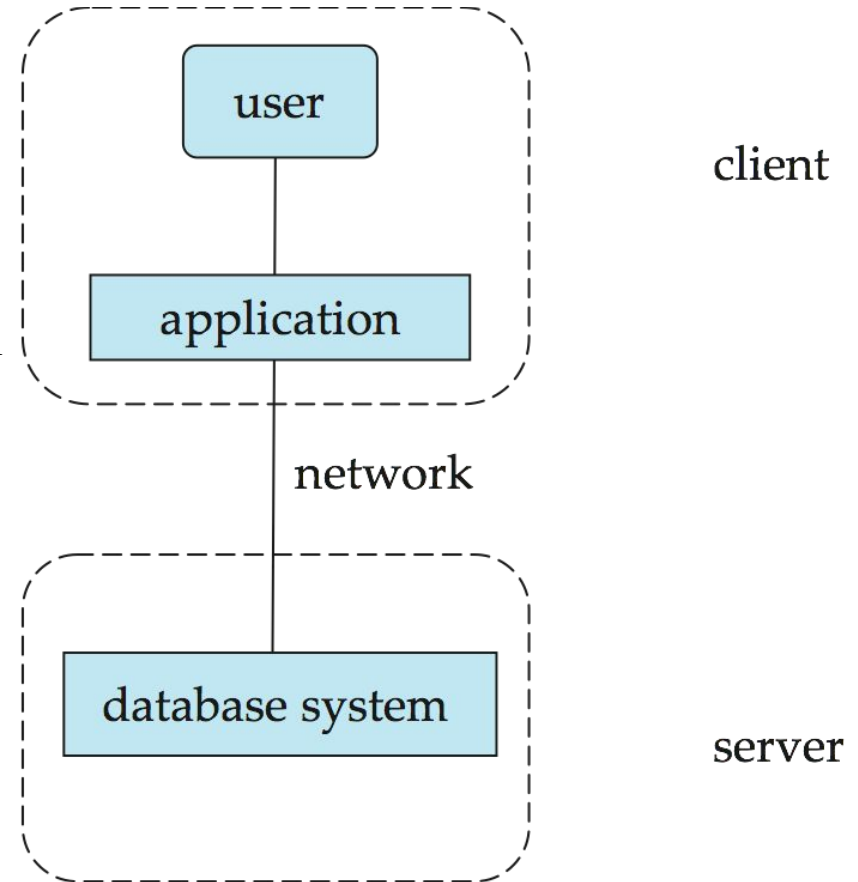






# Client Server Architecture

- Two-tier architecture
  - Must have some application, which uses the DBMS
  - Programmers use 2-tier architecture where they access DBMS by means of application
  - Application tier is entirely independent of database in term of operation, design and programming

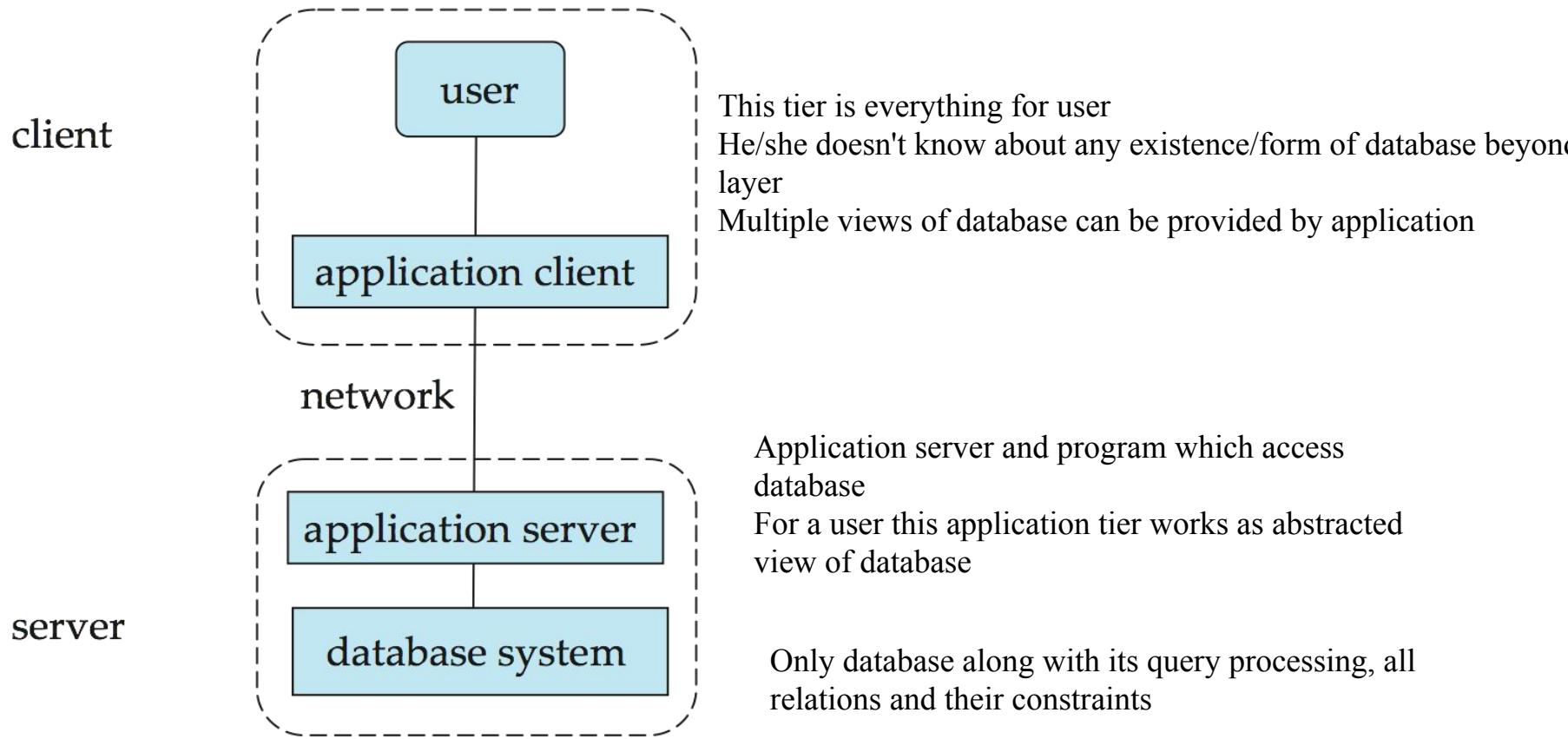


(a) Two-tier architecture



# DBMS Application Architecture

- Three-tier Architecture
  - Most widely used architecture
  - Separates it tier from each other on basis of users



(b) Three-tier architecture



# History of Database Systems

- **Before 1950s :**
  - Data was stored as paper records
  - Lot of man power involved
  - Lot of time was wasted
    - 4 e.g. when searching
  - Inefficient
- **1950s and early 1960s:**
  - Data processing using magnetic tapes for storage
    - 4 Tapes provided only sequential access
  - Punched cards for input, updated new tape will be used as master tape
  - Backup is considered as grand father, father and son relation



# History (cont.)

- **Late 1960s and 1970s:**
  - Hard disks allowed direct access to data
  - Data stored in files, known as File Processing System
    - 4 Network and hierarchical data models in widespread use



# History (cont.)

- **Network data models**

- Data are represented by collections of *records*

- 4 Records and their fields are represented as *record type*

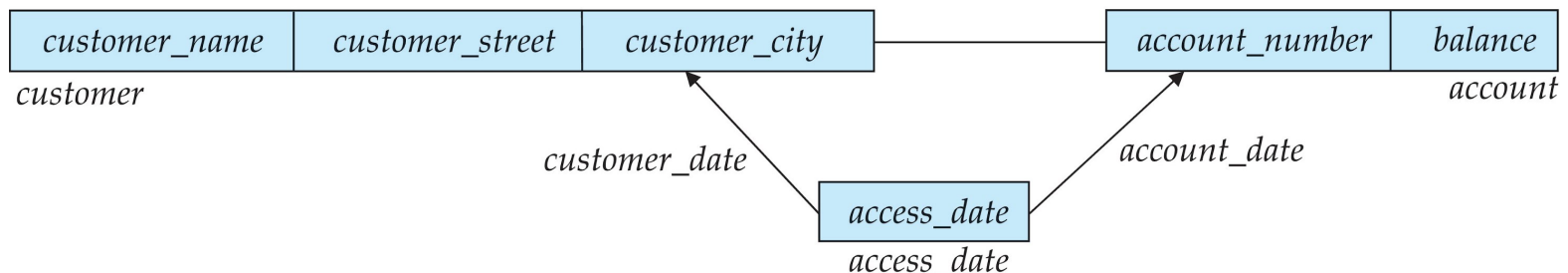
**type** *customer* = **record type** *account* = **record**

*customer-name*: string;      *account-number*: integer;  
*customer-street*: string;      *balance*: integer;  
*customer-city*: string;

**end**      **end**

- Relationships among data are represented by *links*

- 4 Restrictions on links depend on whether the relationship is many-many, many-to-one, or one-to-one

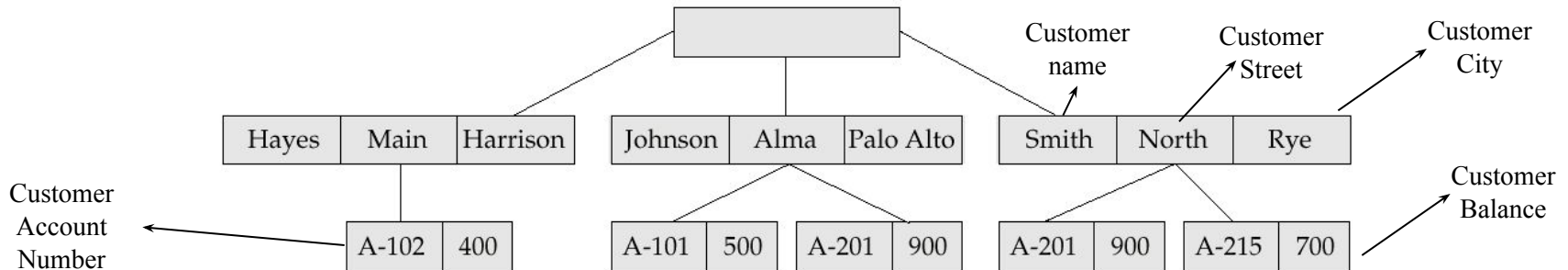




# History (cont.)

- **Hierarchical data models**

- Consists of a collection of *records* which are connected to one another through *links* in the form of a *rooted tree*
  - 4 Record – A collection of fields, each of which contains only one data value
  - 4 Link - An association between precisely two records
- No cycles in the underlying graph
- Relationships formed in the graph must be such that only one-to-many or one-to-one relationships exist between a parent and a child.
- The hierarchical model differs from the network model in that the records are organized as collections of trees rather than as arbitrary graphs





# History (cont.)

- **Late 1960s and 1970s:**
  - Hard disks allowed direct access to data
  - Data stored in files, known as File Processing System
    - 4 Network and hierarchical data models in widespread use
  - Ted Codd defined the relational data model
    - 4 Codd—Mathematician who was working at IBM
    - 4 Convinced that there was a more theoretically rigorous way to represent data on databases and to allow these databases to be accessed without complex programming
    - 4 Thus arose the relational database model, which ushered in the era of RDBMS dominance
    - 4 For a generation of database professionals—more than 25 years—the relational database has reigned supreme
    - 4 IBM Research begins System R prototype
  - High-performance (for the era) transaction processing



# History (cont.)

- **1980s:**
  - Research relational prototypes evolved into commercial systems
    - 4 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
    - 4 Google BigTable, Yahoo PNuts, Amazon, ..





# History (cont.)

- **Current:**
  - Graph Database
  - Columnar Database
  - NoSQL databases, especially those based on Amazon's DynamoDB
    - 4 Explicitly designed to allow the database to continue operating in the presence of a network partition by sacrificing strict consistency
    - 4 Instead, such a database offers eventual consistency
    - 4 Created databases which can only be queried effectively by experienced programmers
  - BDaaS (Big Data as a Service)



# Tools

- Commercial DBMS
  - IBM DB2
  - Oracle
  - Microsoft SQL Server
  - Sybase
  - IBM Informix
- Free DBMS
  - MySQL
  - PostgreSQL



# Topics

## Chapter

1. Introduction

### **Part 1: Relational Databases**

2. Introduction to the Relational Model

3. Introduction to SQL

4. Intermediate SQL

5. Advanced SQL

6. Formal Relational Query Languages

### **Part 2: Database Design**

7. Database Design: The Entity-Relationship Approach

8. Relational Database Design

9. Application Design

### **Part 3: Data Storage and Querying**

10. Storage and File Structure

11. Indexing and Hashing

12. Query Processing

13. Query Optimization

### **Part 4: Transaction Management**

14. Transactions

15. Concurrency Control

16. Recovery System

### **Part 5: System Architecture (Intro)**

17. Database System Architectures

18. Parallel Databases

19. Distributed Databases

### **Part 6: Data Warehousing, Mining, and IR (Intro)**

20. Data Mining

21. Information Retrieval

### **Part 8: Specialty Databases (Intro)**

22. Object-Based Databases

23. XML



# Tutorial Questions

1. Explain the disadvantages of conventional file system.
2. Explain programming language classification with examples of languages.
3. Enlist different data models.
4. Explain the different users of DBMS.
5. Draw the history hierarchy of DBMSs.
6. Explain the concept of physical data independence and its importance in database systems.
7. Two disadvantages associated with database systems.



# End of Chapter 1

## Chapter 2 Relational Model