Back to Dashboard

Purpose of Database Systems

Lecturer: Jane YOU

Compiler: Zhen TONG

In our daily life, we often use file system in our computer operating system to store file, and we have no problems reading, writing, and sharing files. So why cannot we just use file system as a database system? Using a file system as a database system has several limitations and challenges compared to using a dedicated database management system (DBMS). Let's break down these issues:

1. **Data Redundancy and Inconsistency:**
   - In a file system, data is often stored in multiple files, each with its own format and structure. This leads to duplication of information in different files, which can result in data redundancy and inconsistencies.
   - For example, if you have customer information in one file and order information in another, you may duplicate customer details for each order, increasing the likelihood of inconsistent or outdated data.
2. **Difficulty in Accessing Data:**
   - In a file system, you typically need to write a new program or script to perform each new data-related task or query. This can be time-consuming and error-prone, as you need to handle data parsing, file I/O, and data manipulation manually. There's no query language or standard interface for retrieving and managing data in a file system, making it challenging to work with the data efficiently.
   - We have query language like SQL in DBMS to retrieve data.
3. **Data Isolation:**
   - Data is scattered across multiple files and formats, making it difficult to maintain a consistent and organized data structure. Different departments or users might create their own files with varying structures, leading to data isolation and fragmentation. This isolation makes it challenging to establish relationships between different pieces of data and retrieve meaningful insights.
   - For example, there are two products in file system

     -
       ```
       Product Name: Laptop
       Price: $1000
       Quantity in Stock: 10
       Supplier: SupplierA
       ```

     -
       ```
       Product Name: Smartphone
       Price: $500
       Quantity in Stock: 20
       Supplier: SupplierB
       ```

Each product's data is isolated in its own file, making it challenging to analyze overall inventory trends or relationships between products.

- However, if we have a DBMS for products

Product ID	Product Name	Price	Quantity	Supplier
1	Laptop	1000	10	SupplierA
2	Smartphone	500	20	SupplierB

All product data is integrated into a single table, enabling you to analyze inventory trends and relationships more efficiently.

4. **Integrity Problems:**
   - Integrity constraints, such as ensuring that an account balance remains positive, are often buried within program code.
   - These constraints are not explicitly defined within the file system, making it hard to enforce data integrity.
   - Adding new constraints or modifying existing ones can be complex and may require changes to multiple programs, increasing the risk of errors.
   - **Example: Student Enrollment System**

     Suppose you're tasked with creating a student enrollment system for a university. Students can enroll in courses, and there are certain integrity constraints that need to be maintained.

     Integrity Constraint 1: Each student should be assigned a unique student ID.

     Integrity Constraint 2: Students can only enroll in courses that exist in the course catalog.

     Now, let's examine how integrity problems can occur in two scenarios: one using a file system and the other using a database system.

     File System Approach:

     In this approach, you decide to use a file for each student and another file for the course catalog. Here's what your file system might look like:

     - `student1.txt` (Student Record):

```
codeStudent ID: 101
Name: John Doe
Courses Enrolled: Math101, Chemistry201
```

     - `course_catalog.txt` (Course Catalog):

```
Courses Available: Math101, Biology101, Chemistry201
```

     Integrity Problem 1 (Unique Student IDs): In the file system, ensuring unique student IDs is challenging. If you want to add a new student, you must manually check all existing student files to avoid assigning a duplicate ID. If there's a mistake and two students end up with the same ID (e.g., both have ID 101), it violates the integrity constraint.

Integrity Problem 2 (Enrollment in Valid Courses): To ensure that students can only enroll in valid courses, you need to manually check if each course listed in a student's file matches a course in the course catalog. If a student enrolls in a course that doesn't exist in the catalog (e.g., "History101"), it violates the integrity constraint.

Database System Approach:

In contrast, using a database system, you'd have two tables: one for students and another for courses. Your data might look like this:

Students Table:

Student ID	Name
101	John Doe
102	Jane Smith

Courses Table:

Course Code	Course Name
Math101	Mathematics
Biology101	Biology
Chemistry201	Chemistry

Integrity Constraint 1 (Unique Student IDs): In a database, you can easily enforce unique student IDs by defining the "Student ID" column as a primary key. The database system will ensure that no two students share the same ID.

Integrity Constraint 2 (Enrollment in Valid Courses): The database system can enforce referential integrity by setting up a foreign key constraint. This ensures that a student can only enroll in courses that exist in the "Courses" table. If a student tries to enroll in a non-existent course, the database system will prevent it, maintaining data integrity.

5. **Atomicity of Updates:**
   - File systems lack built-in mechanisms to ensure the atomicity of transactions.
   - If a failure occurs during an update operation (e.g., transferring funds between accounts), it may leave the database in an inconsistent state with partial updates.
   - In a database system, transactions are designed to be atomic, ensuring that either the entire operation completes successfully or none of it occurs.
   - **Scenario: Money Transfer Operation**

     Suppose you're developing a banking application, and you need to implement a money transfer from one account to another. You want to ensure that the transfer operation is atomic, meaning it either completes successfully or fails entirely without leaving the system in an inconsistent state.

     File System Approach:

     In the file system approach, you maintain account information in separate text files, and you need to manually implement the money transfer operation. Here's a simplified representation:

- `account1.txt` (Account 1 Information):

  ```
  Account Number: 1001
  Balance: $1,000
  ```

- `account2.txt` (Account 2 Information):

  ```
  Account Number: 1002
  Balance: $500
  ```

To transfer $200 from Account 1 to Account 2:

1. Read `account1.txt` and `account2.txt` to get their balances.
2. Deduct $200 from Account 1 and update `account1.txt`.
3. Add $200 to Account 2 and update `account2.txt`.

Now, imagine a failure occurs (e.g., a power outage or a program crash) after step 2 but before step 3. In this case, `account1.txt` would show a reduced balance, but `account2.txt` wouldn't reflect the money transfer. The system is in an inconsistent state, and the money appears to have disappeared.

Database System Approach:

In a database system, the same money transfer operation is managed within a transaction:

```sql
sqlCopy codeBEGIN TRANSACTION;

-- Step 1: Retrieve account balances
SELECT Balance INTO @Balance1 FROM Accounts WHERE AccountNumber =
1001;
SELECT Balance INTO @Balance2 FROM Accounts WHERE AccountNumber =
1002;

-- Step 2: Deduct $200 from Account 1
UPDATE Accounts SET Balance = @Balance1 - 200 WHERE AccountNumber =
1001;

-- Step 3: Add $200 to Account 2
UPDATE Accounts SET Balance = @Balance2 + 200 WHERE AccountNumber =
1002;

COMMIT TRANSACTION;
```

In a database system:

- The `BEGIN TRANSACTION` statement marks the start of a transaction.
- All the steps are executed within the same transaction.
- If a failure occurs at any point within the transaction (e.g., a power outage or program crash), the database system will automatically roll back the transaction to its initial state (before the `BEGIN TRANSACTION` statement), ensuring that no changes are applied.

6. **Concurrent Access by Multiple Users:**
    - File systems are not well-suited for concurrent access by multiple users or applications.
    - When multiple users attempt to read or modify the same data simultaneously, uncontrolled concurrent accesses can lead to data inconsistencies.
    - Database management systems provide concurrency control mechanisms to handle multiple simultaneous operations.
7. **Security Problems:**
    - Securing data in a file system can be challenging. It's often hard to provide fine-grained access control to different users or groups.
    - A database system offers more robust security features, allowing administrators to define user roles, permissions, and access control policies more effectively.

View of Data

Most of the time, the user of our database system is unprofessional person. Sometimes, even computer programmer can be innocent to the underlying structure of the database, like the front-end programmers.

Data Models

A data model is a conceptual framework or set of tools used to describe data, data relationships, data semantics, and consistency constraints within a database or information system. Data models provide a structured way to represent and organize data, making it easier to understand, manipulate, and manage. Here are examples of data models that address various aspects of data:

1. Relational Model:

- **Description:** The relational model is one of the most widely used data models in database management. It represents data as tables (relations) with rows (tuples) and columns (attributes). Briefly, you can get a first impression for it as a table.
- **Example:** In a relational database, you might have a "Faculty" table with columns for customer ID, name, email, and address. This model defines the relationships between tables using keys.
    - **Table Name:** Faculty
    - **Columns:**
        1. **Faculty ID (Primary Key):** This column stores a unique identifier for each faculty member. It serves as the primary key, ensuring each faculty member has a unique ID within this table.
        2. **Name:** This column stores the full name of the faculty member.
        3. **Email:** It stores the faculty member's email address.
        4. **Address:** This column holds the faculty member's physical address.
    -

Faculty ID	Name	Email	Address
101	John Doe	johndoe@cuhksz.com	DaoYuan 311
102	Jane Smith	janesmith@cuhksz.com	DaoYuan 410

Faculty ID	Name	Email	Address
103	Robert Johnson	robertj@cuhksz.com	DaoYuan 209
...	...	...	...

2. Entity-Relationship (ER) Data Model:

- **Description:** The ER data model is mainly used for database design. It focuses on entities (objects), attributes, and the relationships between entities.
- **Example:** In an ER diagram, you can represent entities like "Customer" and "Order," their attributes, and the relationships between them (e.g., a customer can place multiple orders).
  - **Entities:**
    1. **Customer Entity:** This entity represents customers. It has attributes like "Customer ID," "Name," "Email," and "Address."
    2. **Order Entity:** This entity represents orders placed by customers. It has attributes like "Order ID," "Order Date," and "Total Amount."
  - **Attributes:**
    - Customer Entity Attributes:
      - Customer ID (Primary Key)
      - Name
      - Email
      - Address
    - Order Entity Attributes:
      - Order ID (Primary Key)
      - Order Date
      - Total Amount
  - **Relationships:**
    - A customer can place multiple orders, so there is a "Placed" relationship between the "Customer" and "Order" entities. This is often represented with a connecting line labeled "Placed" and a crow's foot notation on the "Customer" side, indicating the "one" side of a one-to-many relationship, and a straight line on the "Order" side, indicating the "many" side.

3. Object-Based Data Models (Not important in this course):

- **Description:** Object-based data models extend the relational model to support complex data types and object-oriented programming concepts. They allow for the representation of objects with methods and attributes. People taking this course should have a background knowledge of this
- Examples:
  - **Object-Oriented Model:** In an object-oriented database, you can have classes like "Person" with methods and attributes, representing real-world objects.
  - **Object-Relational Model:** Combines relational and object-oriented features, allowing you to define structured data types and methods within a relational database.

4. Semi-Structured Data Model(Not important in this course):

- **Description:** Semi-structured data models are used to represent data that doesn't conform to a strict schema. They allow for flexibility in data representation and are common in scenarios like storing JSON or XML data. People with a web developing background should know JSON, and people with a computer graphics background should have some knowledge of XML
- Examples:
    - **JSON:** JSON (JavaScript Object Notation) is a semi-structured data model used to represent data as key-value pairs, arrays, and nested structures.
    - **XML:** XML (eXtensible Markup Language) provides a hierarchical and semi-structured way to represent data with user-defined tags.

5. Other Older Models(Not important in this course):

- **Network Model:** The network data model represents data as a collection of records connected by links, forming a network-like structure. It was prevalent in the early days of databases.
- **Hierarchical Model:** The hierarchical data model organizes data in a tree-like structure with parent-child relationships. It was commonly used in early database systems, such as IMS (Information Management System).

Level of Abstraction

Levels of abstraction in computer science and information technology refer to the degree of detail or complexity at which a system is represented or viewed. There are typically three primary levels of abstraction. They are connected to each other

1. **View or User Level (Highest Level):**
    - The view or user level is how end-users use a system or software application.
    - It give instruction to the logical level, and get the structured data from the logical level return
2. **Logical Level (Intermediate Level):**
    - The logical level is an abstraction that focuses on how data and processes are organized and structured, independent of the specific hardware or implementation details.
    - It defines the relationships and interactions between different components of a system in a way that is more human-readable and understandable.
    - The logical level receive high-level instruction from the user level, and translate it to operation for physical level. After raw data returned from physical level, it pack the data into user readable structure.
3. **Physical Level (Lowest Level):**
    - The physical level deals with the actual hardware and its components. It is the closest to the physical implementation of a system.
    - At this level, you are concerned with the specific details of how data is stored, transmitted, and processed at the hardware level.

- It receive operation from the logical level, and execute them, and return the raw data to the logical level

Instances and Schemas

People taking this course should have some background in computer programming. If you take GFH in cuhksz, the instances are exactly like Plato's **Material World**, and schemas are like **World of Forms or Ideas**, the more abstract one. And the schemas can be described logically and physically.

- **Logical Schema** is the design of a logical structure of a database, the specific usage for the database
- **Physical Schema** is the underlying physical structure of the database, like the B+ tree.

Languages

The database can be described into two parts: the user-end, and the engine-end. In the user end, SQL is used as a language to create, add, delete, and retrieve data. In the engine-end, many languages can be served as building materials, for example, golang, java, c++, and even python.

SQL

We will teach sql later, and now all you need to know is that sql is a language that allows you to create, add, delete, and retrieve data. In sql, there are some classes of operation (language)

- **Data Definition Language (DDL)**
  - To create a database logical schemas, we can use the DDL

    ```
    create table instructor (
    ID char(5),
    name varchar(20),
    dept_name varchar(20),
    salary numeric(8,2))
    ```

  - This table templates is stored in a data dictionary.
  - The data defining the data is called metadata(data about data), which is stored in data dictionary.
    - Database schema
    - Integrity constraints
      - Primary key (ID uniquely identifies instructors)
    - Authorization
      - Who can access what
- **Data Manipulation Language (DML)**
  - DML is nothing but a tool to access data (fancy name:( )
    - Theoritical(Pure): A mathematical way to optimize the procedure.
    - Practical(Commercial): SQL is the most widely used commercial database language

- We can further divide the language into **Procedural DML** and **Declarative DML**. The former require the knowledge how to get the data, while the later doesn't, which means that the later is more detached the local physical design like the sql.
-

Database Design & Engine

Assume you are a database designer, what work should you do? This question is too ambiguous, because we first need to know what part are you designing. Is it a logical design or a Physical design.

- **logical design** :Programmers pay more attention to the business requirement, which is related to specific commercial behavior. It is about how are you going to design the relationships between entities.
- **physical design**: The engine for the database framework, the database engine.

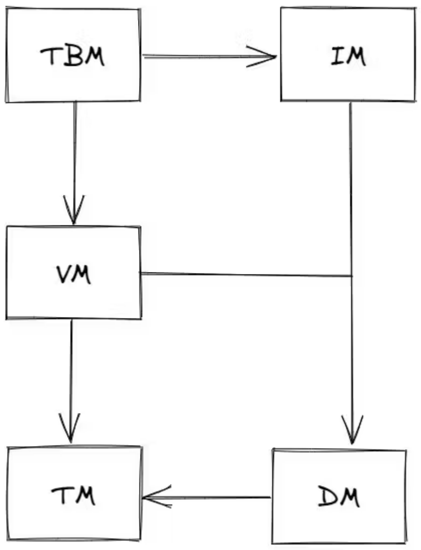
Database Engine

The database system can be divided into :

The storage manager

- The storage manager is the interface between the application programs and low-level data. It needs to interact with the os file manager, so that the os file manager can hand over the data to the query operator.
- The query processor component
- The transaction management component
  - Authorization and integrity manager
  - Transaction manager (TM)
  - File manager
  - Buffer manager
  - Index manager (IM)
- Data in storage manager
  - **Data Files:**
    - Data files are the primary storage units for the database. They store the actual data records, which can include information about customers, products, transactions, and more, depending on the database's purpose. Data files organize data into tables, rows, and columns. Each table corresponds to a specific entity in the database, and rows contain individual data records, while columns represent attributes or properties of those records.
  - **Data Dictionary:**
    - The data dictionary, also known as the system stores metadata about the structure of the database. It contains information about tables, columns, indexes, constraints, and other database objects.
  - **Indices (Indexes):**

- Indices are additional data structures used to optimize data retrieval and query performance. They provide fast access to specific data items within a table. An index consists of a set of keys and pointers. The keys are values from one or more columns of a table, and the pointers point to the actual data records containing those values.
- When a query includes conditions or filters based on indexed columns, the DBMS can use the index to **quickly** locate the relevant data without scanning the entire table.
- Common types of indexes include B-tree indexes, hash indexes, and bitmap indexes, which we will learn later.



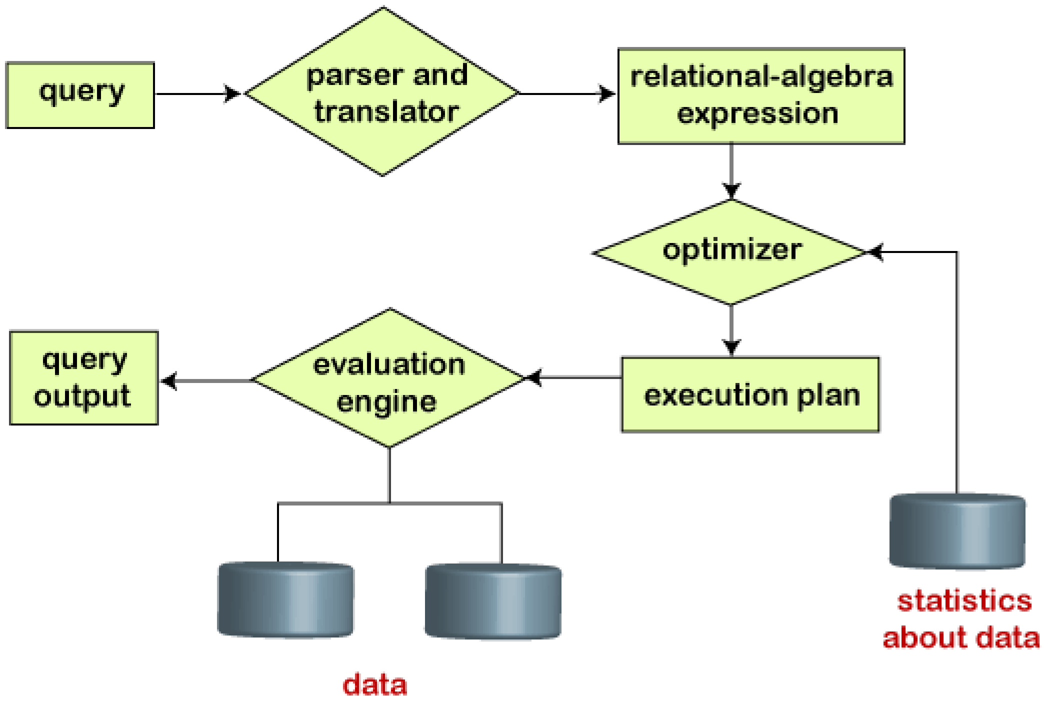
Query Processor

Query processing is a crucial phase in a database management system (DBMS) that involves translating a user's query into a series of actions that retrieve the requested data efficiently from the database. Query processing typically consists of three main steps:

1. **Parsing and Translation:**
    - In this initial step, the DBMS parses the user's query to understand its structure and intension. Parsing involves breaking down the query into its constituent elements, such as keywords, table names, conditions, and operators. (People with a security background know that the parsing procedure is crucial in sql injection attack.)
    - After parsing, the DBMS translates the parsed query into an internal representation that the system can work with. This internal representation often takes the form of a query execution plan or a query tree.
    - The translation phase also involves checking the query's **syntax and semantics for correctness**. Any syntactical or semantic errors are typically identified and reported to the user.
2. **Optimization:**
    - Once the query is parsed and translated, the DBMS aims to optimize the query execution plan to maximize efficiency and minimize resource usage.
    - Query optimization involves selecting the most efficient way to access and retrieve data from the database. The DBMS considers various factors, such as available indexes, join algorithms, and access paths, to determine the optimal execution plan.

- The goal of query optimization is to reduce the query's response time by minimizing the number of disk I/O operations and other resource-intensive tasks.
- The DBMS may use cost-based or rule-based optimization strategies to select the best execution plan.

3. **Evaluation:**
   - In the final step, the DBMS executes the optimized query by following the execution plan generated in the previous step.
   - Data retrieval and processing take place according to the chosen plan. This may involve accessing data from tables, applying filters, joining tables, aggregating results, and sorting data as needed.
   - The DBMS collects and returns the results of the query to the user or application.
   - During evaluation, the DBMS may use techniques such as caching, buffering, and pipelining to improve query performance.



Steps in query processing

Transaction Management (TM)

Transaction management is a fundamental aspect of database management systems (DBMS) that ensures the integrity, consistency, and reliability of data within a database, especially in a multi-user and concurrent access environment. It revolves around the concept of a "transaction," which is a sequence of one or more database operations that together form a single logical unit of work. **Don't worry with this conception nowing, we will teach this in detail later. Just give an impression.** In short, it is dealing with database event in a short period of time.

1. **Definition of a Transaction:**

- A transaction is a set of database operations that are treated as a single, indivisible unit of work. These operations can include inserting, updating, deleting, or querying data in the database.
- Transactions are used to maintain data integrity and consistency by ensuring that a series of related operations either all succeed or all fail. In other words, transactions are atomic, meaning they are treated as an all-or-nothing proposition.

2. **ACID Properties:**
   - Transactions are typically required to adhere to the ACID properties, which stand for:
     - **Atomicity:** Transactions are atomic, ensuring that either all operations within the transaction are completed successfully, or none of them are. There are no partial changes to the database.
     - **Consistency:** Transactions bring the database from one consistent state to another. Consistency constraints are enforced to maintain data integrity.
     - **Isolation:** Transactions are isolated from each other to prevent interference or conflicts. Even when multiple transactions are executing concurrently, their operations should not interfere with each other.
     - **Durability:** Once a transaction is committed (successfully completed), its changes become permanent and are stored safely in the database, even in the face of system failures like power outages or crashes.
3. **Concurrency Control Manager:**
   - In a multi-user environment where multiple transactions may be executed simultaneously, a concurrency control manager is responsible for coordinating the execution of these transactions.
   - Concurrency control mechanisms, such as locks and timestamps, are used to ensure that transactions do not interfere with each other in a way that violates data integrity or consistency.
   - The concurrency control manager enforces isolation among concurrent transactions, making sure that their effects on the database remain as if they had executed serially, even though they may be executed in parallel.
4. **Recovery Manager:**
   - The recovery manager is responsible for ensuring the durability of transactions. It maintains a transaction log that records changes made by transactions before they are committed.
   - In the event of a system failure, the recovery manager can use the transaction log to bring the database back to a consistent state by replaying or undoing transactions as necessary.

Architecture

Centralized databases, client-server systems, parallel databases, and distributed databases are different architectures for organizing and managing databases. Each of these approaches has distinct characteristics and is suitable for specific use cases. Here's an explanation of each:

1. **Centralized Databases:**

   - In a centralized database architecture, all data and database operations are managed on a single central server or system. Your first few homework will based on your local computer which is this design.
2. **Client-Server:**
   - In a client-server architecture, the database server is responsible for storing and managing data, while multiple client machines or devices interact with the server to access and manipulate data. **Your final project could be this architecture!**
   - Characteristics:
     - Separation of concerns between clients (users or applications) and the server.
     - Allows multiple clients to access and share data from a centralized database server.
     - Supports concurrent access and often used in enterprise-level applications.
   - Advantages: Scalability, centralized management, and concurrent access.
   - Disadvantages: Potential bottlenecks if not properly designed, security concerns.
3. **Parallel Databases:**
   - Parallel databases leverage multiple cores, shared memory, and parallel processing to improve data processing performance.
   - Characteristics:
     - Use multiple processors or cores to process data in parallel.
     - Can have various configurations, such as shared-memory, shared-disk, or shared-nothing architectures.
     - Suitable for data-intensive applications that require high processing power.
   - Advantages: Enhanced performance for data-intensive tasks, scalability.
   - Disadvantages: Complexity in setup and management, cost.
4. **Distributed Databases:**
   - Distributed databases are designed to manage data across multiple geographical locations or nodes.
   - Characteristics:
     - Data is distributed across different servers or nodes, which can be in different physical locations.
     - Often used in scenarios where data needs to be accessible from various locations.
     - Can handle schema and data heterogeneity, making it suitable for large-scale and diverse data environments.
   - Advantages: Geographical distribution, fault tolerance, scalability, support for diverse data types.
   - Disadvantages: Complex to manage and coordinate, potential latency due to distributed nature.

2 and 3 Tier architectures

Advantages of 2-Tier Architecture:

- Simplicity: Easier to design and implement for small-scale applications.

- Low latency: Direct communication between the client and server tiers can result in faster response times for certain tasks.

Disadvantages of 2-Tier Architecture:

- Scalability limitations: Limited scalability due to the client handling both the presentation and business logic.
- Security concerns: Security risks associated with exposing business logic on the client-side.

Advantages of 3-Tier Architecture:

- Improved scalability: Separation of business logic and data management allows for better scalability as the load can be distributed across multiple application servers.
- Enhanced security: Business logic is not exposed on the client side, reducing security risks.
- Maintenance and flexibility: Easier to maintain and update individual tiers without affecting others.

Disadvantages of 3-Tier Architecture:

- Increased complexity: More components and interactions make the architecture more complex to design and manage.
- Potential latency: The additional communication layer between the client and server may introduce some latency compared to the 2-Tier model.

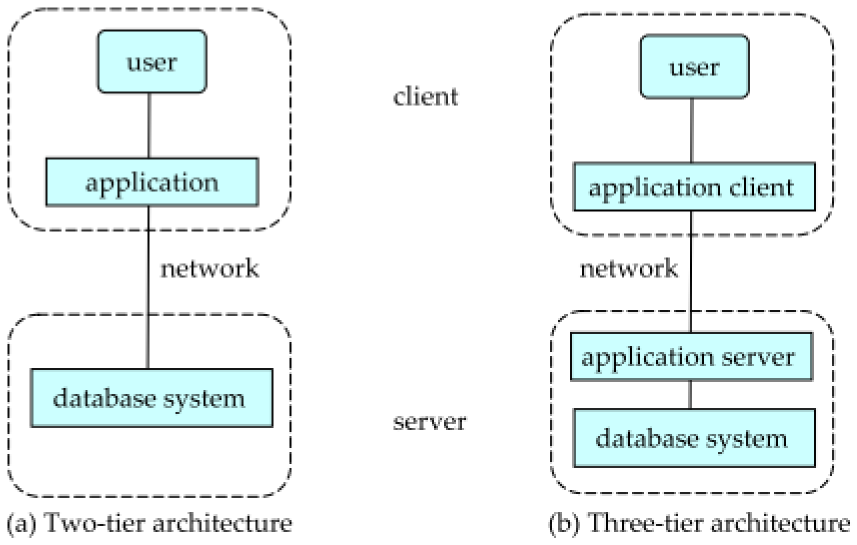


Figure 1.6 Two-tier and three-tier architectures.