Asst. prof. Dr. Ali Hakem Alsaeedi / Dr. Dhiya Abdulsattar Al-khazraji
Computer Information Systems Dept. / Second Grade / First Semester

Approaches to and Evolution of Database Systems

Introduction to Database Systems

Database systems have evolved significantly over the years, from simple file systems to complex and scalable database management systems (DBMS). The development of database systems has aimed to address challenges like data redundancy, inconsistency, and inefficiency. The main approaches and phases in the evolution of database systems include hierarchical, network, relational, and object-oriented systems, with recent trends focusing on NoSQL, big data, and cloud-based databases.

---

1. File Processing Systems (Pre-Database Era)

- **Description**: Before modern databases, organizations used file processing systems, which involved storing data in flat files managed by applications.
- **Limitations**:
    - Data redundancy and inconsistency: Multiple copies of data led to errors and wasted storage.
    - Limited data sharing: Each application accessed its own set of files, hindering integration.
    - Lack of security and data integrity controls.
- **Example**: A payroll system might maintain separate files for employee data and payment records, leading to duplication of employee information.

---

2. Network Database Systems

- **Description**: Developed in the 1970s, network databases allow many-to-many relationships using a graph structure.
- **Features**:
    - More flexibility compared to hierarchical systems.
    - Use of pointers to navigate between records.
- **Limitations**:
    - Complexity: Requires a detailed understanding of the schema to query data.
    - Maintenance challenges due to pointer-based links.
- **Example**: CODASYL (Conference on Data Systems Languages) model databases.

---

3. Relational Database Systems

- **Description**: Introduced in the 1980s by Edgar F. Codd, relational databases organize data into tables (relations) and use Structured Query Language (SQL) for querying.
- **Features**:
    - Data is represented in rows (records) and columns (attributes).
    - Eliminates redundancy through normalization.
    - Provides flexibility and scalability.
- **Advantages**:
    - Simple to use: SQL queries are intuitive.
    - Data integrity: Enforced through constraints like primary keys and foreign keys.
    - Supports ACID (Atomicity, Consistency, Isolation, Durability) properties.
- **Examples**:
    - MySQL: Popular open-source database.
    - Oracle Database: Used for enterprise-scale applications.
    - Use Case: E-commerce platforms manage products, customers, and orders using relational databases.

Components of Database Systems

A **database system** consists of several essential components that work together to store, retrieve, and manage data efficiently. These components include hardware, software, data, users, and procedures, all of which are integrated to provide a robust system for data management.

---

1. Hardware

- **Description**: The physical devices required for the functioning of the database system. This includes servers, storage devices, and networking hardware.
- **Components**:
    - **Servers**: High-performance computers to store and process database queries.
    - **Storage Devices**: Hard drives, SSDs, or cloud storage where the database is stored.
    - **Networking Equipment**: Facilitates access to the database over local or wide-area networks.

---

2. Software

- **Description**: The set of programs and utilities that enable database creation, management, and operation.
- **Components**:
    1. **Database Management System (DBMS)**:
        - Software that manages data and provides an interface for users and applications.
        - Examples: MySQL, Oracle Database, Microsoft SQL Server.
    2. **Application Programs**:
        - Custom software to interact with the DBMS for specific tasks, such as inventory management.
    3. **Operating System**:
        - The underlying system software that runs the DBMS and related programs.
        - Example: Linux or Windows Server.

---

3. Data

- **Description**: The most critical component, representing the raw facts stored in the database.
- **Types of Data**:
    1. **User Data**: Actual information stored, such as customer names, product details, or transaction records.
    2. **Metadata**: Data about the data, describing structure, constraints, and relationships (e.g., schema).
    3. **Indexes**: Data structures that speed up search operations.

---

4. Users

- **Description**: Individuals or systems that interact with the database, classified into different roles:
    1. **End Users**:
        - Use the database for querying, reporting, or updating data.
        - Example: A sales representative retrieving customer purchase history.
    2. **Database Administrators (DBAs)**:
        - Manage and maintain the database system, ensuring data security, backup, and performance optimization.
    3. **System Analysts and Developers**:
        - Design and implement database-driven applications.

---

5. Procedures

- **Description**: Guidelines and instructions for using and maintaining the database system.
- **Components**:
    1. **Operating Procedures**: Daily operations like data backups, recovery, and access monitoring.
    2. **Development Procedures**: Guidelines for creating and testing new database functionalities.
    3. **Security Procedures**: Policies to control access and protect sensitive data.
- **Example**:
    - A financial institution may implement strict access controls and audit procedures to comply with regulatory requirements.

---

6. Query and Reporting Tools

- **Description**: Interfaces and utilities for interacting with the database.
- **Components**:
    1. **Query Language**:
        - Allows users to retrieve and manipulate data.
        - Example: SQL (Structured Query Language).
    2. **Reporting Tools**:
        - Generate structured reports from database data.
        - Examples: Crystal Reports, Tableau.

---

7. Security and Integrity

- **Description**: Ensures that the database remains protected and consistent.
- **Components**:
    1. **Authentication**: Verifying user identity.
    2. **Authorization**: Controlling user access to specific data.
    3. **Data Integrity**: Enforcing rules to ensure data accuracy and consistency.

Database Architecture and Data Independence

---

1. Database Architecture Overview

Database architecture refers to the design, structure, and organization of the components of a database system. It defines how data is stored, accessed, and managed while ensuring efficiency, scalability, and security. The architecture is typically divided into **three levels**: internal, conceptual, and external. These levels aim to provide a separation between users and the physical data, enabling flexibility and scalability.

1.1 Three-Tier Architecture

1. **Internal Level (Physical Level)**:
    - **Description**: The lowest level of the architecture, dealing with physical data storage.
    - **Components**:
        - Storage media (e.g., hard drives, SSDs, cloud storage).
        - Data structures (e.g., indexes, files, pages).
    - **Responsibilities**:
        - Managing data storage and retrieval.
        - Optimizing storage space and access speed.
    - **Example**: A DBMS stores customer transaction data as binary files on a disk, using indexes for faster retrieval.
2. **Conceptual Level (Logical Level)**:
    - **Description**: Represents the logical structure of the database, defining relationships, constraints, and schemas.
    - **Components**:
        - Database schema: Defines tables, fields, and relationships.
        - Constraints: Enforce data integrity (e.g., primary keys, foreign keys).
    - **Responsibilities**:
        - Ensures logical consistency and data integrity.
        - Abstracts the physical data for application use.
    - **Example**: A relational database schema includes tables for customers, orders, and products, with foreign key relationships linking them.

3. **External Level (View Level)**:
    - **Description**: The highest level, providing tailored views of the database for different users or applications.
    - **Components**:
        - User-defined views: Filter data based on specific requirements.
        - Security rules: Limit access to sensitive data.
    - **Responsibilities**:
        - Enhances usability by presenting only relevant data.
        - Implements security by restricting access.
    - **Example**: An e-commerce platform may present a simplified view of order data to customers, while administrators access detailed sales reports.

1.2 Two-Tier Architecture

- In simpler database systems, the architecture may consist of:
    - A **Client Tier**: End-user interface for accessing data.
    - A **Server Tier**: Centralized database storage and management.
- **Example**: A small business using a desktop database application like Microsoft Access.

---

2. Data Independence

Data independence is the ability to change the database structure at one level without affecting the other levels. It ensures that the database can evolve over time without disrupting applications or user operations.

2.1 Types of Data Independence

1. **Logical Data Independence**:
    - **Definition**: The ability to modify the logical schema (conceptual level) without altering the external schema or application programs.
    - **Example**: Adding a new column to a table or changing a relationship in the schema without affecting user views or queries.
    - **Benefit**: Facilitates database evolution and adaptation to new requirements.
2. **Physical Data Independence**:

- **Definition**: The ability to change the physical schema (internal level) without affecting the logical schema or application programs.
- **Example**: Moving data to a faster storage device or reorganizing file structures without affecting how data is accessed or queried.
- **Benefit**: Enables performance optimization without disrupting operations.

---

3. Importance of Database Architecture and Data Independence

1. **Simplifies Database Management**:
   - Clear separation of levels makes maintenance, optimization, and scaling easier.
   - Example: A company can migrate to a cloud database without altering user queries or applications.
2. **Enhances Security**:
   - The external level allows for restricted access, ensuring sensitive data is only visible to authorized users.
   - Example: Payroll systems show only relevant employee details to HR personnel.
3. **Improves Flexibility**:
   - Logical and physical independence allow businesses to adapt databases to new requirements or technologies without significant disruptions.
   - Example: Retailers can add a new table for online sales tracking without affecting existing systems.
4. **Supports Scalability**:
   - Well-architected databases can scale to accommodate growing data volumes and user demands.
   - Example: A social media platform can optimize physical storage to handle millions of user interactions.

---

4. Examples of Database Architecture in Practice

- **Banking Systems**:
  - Internal Level: Stores encrypted customer transaction data on distributed servers.
  - Conceptual Level: Defines relationships between accounts, customers, and branches.
  - External Level: Provides account statements to customers via online banking interfaces.

- **E-Commerce Platforms**:
    - Internal Level: Uses cloud-based storage to manage product inventories.
    - Conceptual Level: Links products, orders, and users in a relational schema.
    - External Level: Offers personalized views of products for customers and dashboards for administrators.

---

5. Future Trends in Database Architecture

1. **Distributed Architectures**:
    - Data stored across multiple locations for faster access and higher availability.
    - Example: Google Spanner, a globally distributed database.
2. **Cloud Databases**:
    - Databases hosted on cloud platforms with seamless scaling and management.
    - Example: Amazon RDS and Microsoft Azure SQL Database.
3. **Decoupled Architectures**:
    - Using data lakes for raw data storage and separate processing engines for analytics.
    - Example: A retail chain uses a data lake for customer data and a relational database for transactional processing.

Design of Core DBMS Functions

A **Database Management System (DBMS)** is designed with a set of core functions to efficiently manage, retrieve, and update data while ensuring consistency, security, and high performance. These functions include **query mechanisms**, **transaction management**, **buffer management**, and **access methods**, among others.

---

1. Query Mechanisms

Query mechanisms are the tools and processes through which users and applications retrieve or manipulate data in a database.

1.1 Query Processing

- **Steps**:
    1. **Parsing**:
        - The DBMS parses the query to check for syntax errors and translates it into an internal representation.
        - Example: Translating an SQL query (SELECT * FROM employees WHERE department='HR';) into an execution plan.
    2. **Optimization**:
        - The DBMS determines the most efficient way to execute the query by considering factors like indexes and join methods.
        - Example: Choosing between a table scan and an index lookup based on the query conditions.
    3. **Execution**:
        - The query is executed, and results are fetched.

1.2 Query Languages

- **Structured Query Language (SQL)**: The standard language for relational databases.
    - Example: A sales manager querying for monthly sales totals using:

Sql :Example code

```
SELECT SUM(amount) FROM sales WHERE month='October';
```

Example Use Case:

An e-commerce platform uses a query mechanism to fetch product details for a customer search (SELECT * FROM products WHERE name LIKE '%laptop%';).

---

2. Transaction Management

Transaction management ensures that database operations are executed reliably and maintain consistency, even in the presence of failures.

2.1 ACID Properties

- **Atomicity**: A transaction is treated as a single unit; either all its operations succeed, or none do.
    - Example: If a bank transfers $100 from Account A to Account B, both debit and credit operations must complete, or neither should.
- **Consistency**: A transaction brings the database from one valid state to another.
    - Example: Updating inventory levels in an online store after a purchase.
- **Isolation**: Transactions do not interfere with each other.
    - Example: Two customers placing orders simultaneously do not see partial data changes.
- **Durability**: Once a transaction is committed, changes are permanent, even after a system crash.
    - Example: Saving a new customer record in a CRM system ensures it remains available after a power outage.

2.2 Concurrency Control

- **Description**: Ensures multiple transactions can execute simultaneously without leading to conflicts.
- **Techniques**:
    - **Locks**: Prevent conflicting access (e.g., read-write locks).
    - **Timestamps**: Ensure transactions are processed in a consistent order.
    - **Example**: A ticket booking system uses locking to prevent double-booking of seats.

---

3. Buffer Management

Buffer management deals with the efficient use of main memory to store data that is frequently accessed or being processed.

3.1 Role of Buffer Manager

- **Description**: The buffer manager temporarily stores data blocks in memory to reduce disk I/O operations, improving performance.
- **Processes**:
    - **Caching**: Frequently accessed data is stored in a buffer.
    - **Page Replacement Policies**:

  - **Least Recently Used (LRU)**: Replaces the least recently accessed page.
  - **First-In-First-Out (FIFO)**: Replaces the oldest page in the buffer.
- **Example**: A DBMS caches a table's index in memory to speed up search operations for queries.

3.2 Example Use Case:

A bank's ATM system uses buffer management to keep recent transaction logs in memory for fast retrieval.

---

4. Access Methods

Access methods define how the DBMS retrieves data from the storage system efficiently.

4.1 Indexing

- **Description**: An index is a data structure that improves the speed of data retrieval operations.
- **Types**:
  - **B-Tree Index**: Balanced tree structure for efficient range queries.
  - **Hash Index**: Uses hash functions for fast lookups.
- **Example**: A customer database uses a B-tree index on the "customer_id" column to quickly fetch customer details.

4.2 Sequential Access

- **Description**: Data is read sequentially, often used for operations like full table scans.
- **Example**: Generating a report that aggregates sales data for the entire year.

4.3 Random Access

- **Description**: Data is accessed directly using pointers or keys.
- **Example**: Retrieving a specific order from an "orders" table using an order ID.

---

5. Additional Core Functions

5.1 Recovery Management

- Ensures data integrity in case of system failures by using logs and backups.
- **Example**: A banking application rolls back incomplete transactions during a system crash to prevent data corruption.

5.2 Security Management

- Provides mechanisms for authentication, authorization, and data encryption.

- **Example**: Restricting access to sensitive employee salary data to authorized HR personnel.

5.3 Data Storage Management

- Manages how data is physically stored on disks, including compression and partitioning.
- **Example**: A data warehouse partitions sales data by year for faster analytics.

Use of a Declarative Query Language

---

1. Definition and Purpose of a Declarative Query Language

A **declarative query language** allows users to specify *what* data they want to retrieve or manipulate without needing to define *how* to achieve it. These languages abstract the technical details of data access, focusing on the desired result rather than the process.

- **Key Features**:
    1. Focus on *what* rather than *how*.
    2. Simplifies database interactions for users.
    3. Enables the DBMS to optimize query execution automatically.
- **Example**: SQL (Structured Query Language) is the most common declarative query language used in relational databases.

---

2. Characteristics of Declarative Query Languages

1. **Simplicity**:
    - Intuitive syntax reduces the learning curve for users.
    - Example: To retrieve all employees in a company:

Sql: Example code

```
SELECT * FROM employees;
```

2. **Optimization by the DBMS**:
    - The DBMS determines the best way to execute the query.
    - Example: A query involving a large dataset may use an index for efficiency.
3. **High-Level Abstraction**:
    - Users work with logical representations of data (e.g., tables) rather than physical data structures.

---

3. SQL: The Standard Declarative Query Language

SQL is a domain-specific language widely used for managing and querying relational databases. It comprises different types of statements:

3.1 Data Retrieval (SELECT Queries)

- **Description**: Used to fetch data from one or more tables.
- **Example**: Retrieve all customers from a specific city:

Sql: Example code

```
SELECT * FROM customers WHERE city = 'New York';
```

3.2 Data Manipulation

- **Description**: Involves inserting, updating, or deleting data.
- **Examples**:
    - Insert new customer data:

Sql: Example code

```
INSERT INTO customers (name, city) VALUES ('John Doe', 'Boston');
```

    - Update customer city:

Sql: Example code

```
UPDATE customers SET city = 'Chicago' WHERE name = 'John Doe';
```

    - Delete customer record:

Sql Example code

```
DELETE FROM customers WHERE name = 'John Doe';
```

3.3 Data Definition

- **Description**: Used to define or modify database structures.
- **Examples**:
    - Create a new table:

Sql Example code

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  department VARCHAR(50)
);
```

    - Alter a table to add a column:

Sql Example code

```
ALTER TABLE employees ADD COLUMN salary DECIMAL(10, 2);
```

3.4 Data Control

- **Description**: Controls access to data and database objects.
- **Examples**:
    - Grant access to a user:

Sql Example code

GRANT SELECT, INSERT ON employees TO 'user1';

    - Revoke access from a user:

Sql Example code

REVOKE INSERT ON employees FROM 'user1';

---

4. Benefits of Using Declarative Query Languages

1. **Ease of Use**:
    - Non-technical users can interact with databases using simple commands.
    - Example: A sales manager retrieves monthly sales data without needing programming expertise:

Sql Example code

SELECT SUM(amount) FROM sales WHERE month = 'October';

2. **Reduced Complexity**:
    - The DBMS handles query optimization and execution details.
    - Example: A JOIN query across multiple tables is optimized internally:

Sql Example code

SELECT orders.id, customers.name

FROM orders

JOIN customers ON orders.customer_id = customers.id;

3. **Improved Productivity**:
    - Declarative queries are concise, reducing the time required to write and debug code.
4. **Portability**:
    - Queries written in SQL can be executed across multiple relational database systems with minimal changes.

---

5. Use Cases of Declarative Query Languages

1. **E-Commerce**:
    - Retrieve product information based on user search criteria:

Sql Example code

```
SELECT * FROM products WHERE name LIKE '%laptop%';
```

2. **Healthcare**:
    - Find patients admitted in a specific month:

Sql Example code

```
SELECT * FROM patients WHERE admission_date BETWEEN '2024-01-01' AND '2024-01-31';
```

3. **Banking**:
    - Calculate total deposits in a branch:

Sql Example code

```
SELECT SUM(amount) FROM transactions WHERE branch_id = 101 AND type = 'deposit';
```

4. **Education**:
    - List students enrolled in a specific course:

Sql Example code

```
SELECT students.name
FROM students
JOIN enrollments ON students.id = enrollments.student_id
WHERE enrollments.course_id = 'CS101';
```

---

6. Declarative vs. Procedural Query Languages

Aspect	**Declarative (e.g., SQL)**	**Procedural (e.g., PL/SQL)**
Focus	Specifies *what* data to retrieve or manipulate.	Specifies *how* to retrieve or manipulate data.
Complexity	High-level and simple.	Detailed and requires more programming expertise.

Aspect	Declarative (e.g., SQL)	Procedural (e.g., PL/SQL)
Optimization	Handled automatically by the DBMS.	Developer must optimize logic.
Use Case	General querying and reporting tasks.	Complex logic requiring loops and conditions.
Example	SELECT * FROM orders WHERE status = 'shipped';	Use of procedural blocks for batch processing tasks.

Systems Supporting Structured and/or Stream Content

---

1. Overview

Modern organizations manage vast amounts of data that can be classified into two main types: **structured content** and **stream content**. These types of content require different systems and technologies for efficient storage, processing, and analysis.

- **Structured Content**: Organized data stored in fixed formats, such as rows and columns in relational databases.
- **Stream Content**: Continuous flow of real-time data, often unstructured or semi-structured, generated from sources like sensors, social media, and live applications.

---

2. Systems for Structured Content

Structured content is typically managed by traditional **Relational Database Management Systems (RDBMS)** and enterprise systems designed to process organized data.

2.1 Features

- Data stored in predefined schemas (e.g., tables, rows, and columns).
- Supports SQL for querying and manipulating data.
- Best suited for transactional and analytical processes.

2.2 Examples

1. **Enterprise Resource Planning (ERP) Systems**:
    - Manage structured data such as inventory levels, sales, and payroll.
    - **Example**: SAP or Oracle ERP systems use relational databases to manage structured business processes.
2. **Customer Relationship Management (CRM) Systems**:
    - Handle customer data like contact details, purchase history, and interactions.
    - **Example**: Salesforce uses structured content to analyze customer trends.
3. **Business Intelligence (BI) Tools**:
    - Extract structured data from databases for reporting and decision-making.
    - **Example**: Power BI or Tableau visualizes sales data from an SQL database.

---

3. Systems for Stream Content

Stream content involves data that is generated and processed in real-time, often in large volumes and at high velocity. Systems designed for stream content must handle variability and process data on the fly.

3.1 Features

- Processes unstructured or semi-structured data like logs, IoT sensor readings, or video streams.
- Uses distributed architectures to manage high-velocity data.
- Focus on real-time analytics and decision-making.

3.2 Technologies

1. **Stream Processing Platforms**:
    - Process data in motion for immediate analysis.
    - **Example**: Apache Kafka streams real-time financial transactions for fraud detection.
2. **NoSQL Databases**:
    - Handle semi-structured data from streaming sources.
    - **Example**: MongoDB or Cassandra stores data from IoT devices.
3. **Cloud-Based Stream Analytics**:
    - Scalable platforms for real-time data insights.
    - **Example**: Amazon Kinesis processes live video feeds for content moderation.

3.3 Use Cases

1. **Smart Cities**:
    - IoT sensors collect and stream traffic data to optimize traffic lights and reduce congestion.
    - **Technology**: Apache Flink for stream analytics.
2. **Stock Market Monitoring**:
    - Real-time trading systems process financial data streams for algorithmic trading.
    - **Technology**: Stream processing with Apache Storm.
3. **Social Media Monitoring**:
    - Analyzing live tweets and posts for sentiment analysis or trending topics.
    - **Technology**: Elasticsearch processes streaming data from Twitter.

4. Systems Supporting Both Structured and Stream Content

Many modern systems are designed to handle both structured and stream content simultaneously, allowing organizations to integrate historical and real-time data for comprehensive insights.

4.1 Data Warehouses with Streaming Extensions

- **Description**: Traditional data warehouses enhanced with stream processing capabilities.
- **Example**: Google BigQuery allows querying both batch and streaming data.

4.2 Unified Analytics Platforms

- **Description**: Platforms that process structured data from databases and stream content from real-time sources.
- **Example**: Apache Spark integrates batch processing for structured data and stream processing for real-time events.

4.3 Use Cases

1. **E-Commerce**:
    - Structured Content: Historical sales data.
    - Stream Content: Real-time user interactions and clicks.
    - Unified System: Recommendations are generated by combining historical purchase patterns and live browsing behavior.
2. **Healthcare**:
    - Structured Content: Patient records in electronic health systems.
    - Stream Content: Real-time data from wearable health devices.
    - Unified System: Alerts are triggered for abnormal vitals by analyzing streaming data against patient history.

---

5. Challenges and Solutions

1. **Integration**:
    - Challenge: Combining structured and stream content for analysis.
    - Solution: Use hybrid systems like Apache Spark or cloud-native solutions like AWS Lambda.
2. **Scalability**:
    - Challenge: Managing large-scale, high-velocity data streams.
    - Solution: Distributed architectures with horizontal scaling (e.g., Kafka clusters).

3. **Data Consistency**:
    - Challenge: Ensuring consistency between structured and stream data.
    - Solution: Event-driven architectures and ACID-compliant databases.
4. **Real-Time Analytics**:
    - Challenge: Processing and analyzing data in real-time.
    - Solution: Stream processing frameworks like Apache Flink or Google Dataflow.

---

6. Future Trends

1. **Edge Computing**:
    - Processing stream content closer to the source for low-latency applications.
    - Example: Autonomous vehicles process sensor data on the edge for immediate decisions.
2. **AI Integration**:
    - Leveraging AI to analyze both structured and stream data for predictive analytics.
    - Example: AI-powered fraud detection systems in banking.
3. **Hybrid Cloud Architectures**:
    - Combining on-premises structured data systems with cloud-based streaming services.
    - Example: A manufacturing company uses AWS IoT Core for live machine data and on-prem SQL for production records.
