DBMS

Week 09

Servlets

Servlet API

- The Java Servlet specification defines an API (Application Programming Interface) for communication between web or application servers and Java application programs. Servlets are Java classes that implement this API and are used to handle HTTP requests and generate responses dynamically.

HTTP Servlet Class

- The HTTP Servlet class in Java implements the servlet API specification. Developers create specific servlet classes by extending HTTP Servlet and implementing its methods. These servlet classes are used to define the behavior for handling various types of HTTP requests (e.g., GET, POST).

Servlet Example

- An example of a servlet named PersonQueryServlet. This servlet processes an HTTP GET request, extracts parameters sent from an HTML form, and performs a database query based on those parameters. The results are dynamically generated as an HTML response and sent back to the client's browser.

Servlet Loading

- Servlets are loaded into the web or application server when the server starts or when it receives an HTTP request that requires the servlet's execution. Each servlet request is typically handled in its own thread, allowing multiple requests to be processed concurrently.

Session Management

- Servlets can maintain session information to track user interactions and maintain user state across multiple requests. This is often achieved through the use of cookies, session objects, and HTTP Session.

Servlet Life Cycle

- Servlets have a well-defined life cycle controlled by the server. When a servlet is first accessed, it's loaded, initialized (using the init method), and can then handle multiple requests (using the service method). Servlet instances can also be destroyed when they are no longer needed (using the destroy method).

Application Servers

- Application servers, such as Apache Tomcat, Glassfish, JBoss, and others, provide built-in support for servlets. These servers manage the deployment, execution, and monitoring of servlet-based applications. They may offer additional services and support for Java EE (Enterprise Edition) features.

Integrated Development Environments (IDEs)

- Developers typically use integrated development environments like Eclipse or NetBeans to develop servlet-based applications. These IDEs often come bundled with servlet containers, making it easy to develop, test, and deploy servlets.

Alternative Server-Side Frameworks

Server-Side Scripting

- Server-side scripting involves using scripting languages to embed code within HTML documents. These scripts are executed on the server before delivering the web page to the client's browser.
- Server-side scripts can generate dynamic content, interact with databases, and manipulate the HTML content of web pages.
- Examples of server-side scripting languages include Java Server Pages (JSP), ASP.NET, PHP, and Ruby on Rails.

Java Server Pages (JSP)

- JSP allows developers to embed Java code within HTML pages. The embedded Java code generates the dynamic parts of web pages.
- JSP pages are translated into servlet code, compiled, and executed by the application server.
- JSP also supports tag libraries and provides tools for simplifying web development tasks, such as creating forms and handling database access.

PHP

- PHP is a popular server-side scripting language that can be mixed with HTML code.
- PHP code is enclosed within <?php ... ?> tags and can be used to generate dynamic content and interact with web forms.
- PHP has extensive libraries and support for database access.

Web Application Frameworks

- Web application frameworks provide a higher-level abstraction for web development. They offer a set of tools and features to simplify web application development.
- Features of web application frameworks may include URL routing, form handling, user authentication, object-relational mapping (ORM), and more.
- Examples of web application frameworks include Django (Python), Ruby on Rails (Ruby), Apache Struts (Java), and more.

The Django Framework

- Django is a web application framework for Python. It provides features like URL mapping, views, and templates.
- Views in Django are similar to servlets in Java, and they handle HTTP requests and generate responses.
- Django simplifies tasks like user authentication, form handling, and database access through its built-in features and libraries.

Client-Side Code and Web Services

JavaScript (JS)

- JavaScript is a widely used programming language primarily employed for creating interactive and dynamic elements on web pages. It is a client-side scripting language, meaning it runs in a user's web browser and can manipulate web page content in real-time. JavaScript allows web developers to enhance user interfaces, validate form data, make asynchronous requests to web servers (Ajax), and create responsive web applications. It is a core technology for modern web development.

Web Services

- Web services are software components or applications that provide a standardized way for different software systems to communicate and exchange data over the internet. They use established protocols, such as HTTP, to enable interoperability between various platforms and programming languages. Web services can perform various functions, including data retrieval, data storage, data processing, and more. Common formats for exchanging data in web services include JSON and XML. Web services are used to enable different applications or systems to work together seamlessly.

Mobile Application Development

- Mobile application development refers to the process of creating software applications specifically designed to run on mobile devices, such as smartphones and tablets. These applications, commonly known as mobile apps, leverage the unique capabilities of mobile devices, including touch screens, cameras, GPS, and sensors. Mobile app development typically involves programming languages and frameworks tailored to specific mobile platforms, such as Android or iOS. Developers can create native apps (platform-specific) or cross-platform apps (compatible with multiple platforms) based on their requirements and target audience.

Application Architectures

Web Application Layers

- Presentation/User-Interface Layer: Responsible for user interaction and can have multiple versions for different types of interfaces (e.g., web browsers, mobile apps). Often structured using the MVC architecture.
- Business-Logic Layer: Provides a high-level view of data and actions on data. It includes abstractions of entities and workflows, ensuring that business rules are satisfied.
- Data-Access Layer: Acts as an interface between the business-logic layer and the underlying database. It handles mapping between the object-oriented data model used by the business logic and the relational model used by the database.

Business-Logic Layer

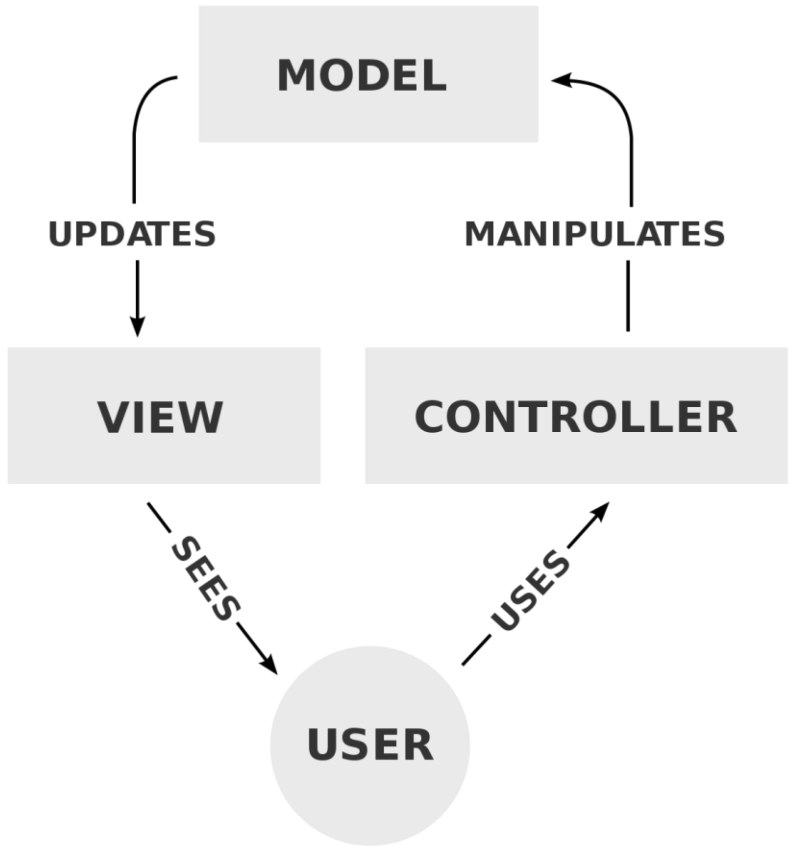
- This layer handles high-level business logic, including abstractions of entities, actions on data, and workflows. It enforces business rules, such as prerequisites for student enrollment in a university.

Data-Access Layer and Object-Relational Mapping (ORM)

- The data-access layer provides an interface between the object-oriented business-logic layer and the relational database.
- ORM frameworks like Hibernate and Django ORM automate the mapping between objects and relational data, allowing developers to work with objects in their code while handling database operations transparently.
- ORM frameworks generate SQL queries and handle the conversion of data between objects and database tables.

Model-View-Controller (MVC) Architecture

- Model: Corresponds to the business-logic layer and represents entities and business logic. It manages data and enforces business rules.
- View: Defines the presentation of data and can have multiple versions tailored to specific devices. It generates the user interface.
- Controller: Receives user events, executes actions on the model, and returns views to the user. It manages the flow of data between the model and view.



Example ORM Tools

- Hibernate (Java): Provides Java Persistence API (JPA) implementation for mapping Java objects to database tables. It simplifies database operations and supports object-oriented data models.
- Django ORM (Python): Part of the Django web framework, it offers an ORM for Python developers, allowing them to define models in Python classes and interact with the database using Python code.

Application Performance

Connection Pooling

- To reduce overhead when connecting to a database, connection pooling is used. Instead of opening a new database connection for each user request, a connection pool manager creates a pool of open connections. The application code requests a connection from the pool and returns it when done. If no available connections are in the pool, a new one may be created (within limits). This reduces the time and resources required to establish a new database connection.

Caching

- Caching is a crucial technique to speed up web application performance. It involves storing and reusing previously fetched data or web pages to avoid redundant queries or computations. Caching can occur at various levels:
  - Query Result Caching: Storing the results of database queries and reusing them if the same query is requested again without changes to the data. This reduces the need for repeated database access.
  - Web Page Caching: Storing entire web pages and reusing them when the same request is made, avoiding the need to regenerate the page content.
  - Materialized Views: Caching data that represents a snapshot of the database, which needs to be invalidated or refreshed when the underlying data changes.

Main Memory Caching

- Main memory caching systems like Memcached and Redis are used to store frequently accessed data in RAM. These systems allow applications to cache data by associating it with a key, enabling fast data retrieval without hitting the database. Cached data can be invalidated or updated manually when needed.

Parallel Processing

- To handle a high volume of requests, web applications can employ parallel processing by using multiple application servers running in parallel. Load balancing mechanisms distribute incoming requests across these servers, ensuring that all requests from a single client session are routed to the same server (to maintain session state). This approach helps distribute the processing load and prevents a single server from becoming a bottleneck.

Database Scaling

- To prevent the database from becoming a bottleneck, database scaling techniques are employed. Parallel database systems are used to handle large amounts of data and queries efficiently. Additionally, parallel data storage systems accessible via web service APIs can be utilized to manage extensive user data.

Application Security

Authentication

- Applications must authenticate users to ensure that only authorized users can access the system. Passwords are a common form of authentication, but more robust methods like two-factor authentication (2FA) are also used.

SQL Injection

- SQL injection is a common security vulnerability where attackers manipulate user inputs to execute malicious SQL queries. Prepared statements or input validation can help prevent SQL injection.

Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF)

- XSS and CSRF attacks involve injecting malicious scripts or making unauthorized requests through a user's browser. Preventive measures include input validation and checking the referrer and request origin.

Password Leakage

- Storing passwords in clear text within application code is a security risk. It's recommended to store passwords securely, possibly using encryption, and restrict database access to trusted sources.

Application-Level Authentication

- Applications often implement their authentication systems, which can include two-factor authentication or integrating with central authentication services like LDAP or SAML for single sign-on.
- While SQL provides authorization mechanisms, application-level authorization is necessary for fine-grained control, such as restricting access to specific data for different users. Row-level authorization mechanisms can be employed.

Audit Trails

- Audit trails log all changes to application data, helping in identifying security breaches or erroneous updates. They can also assist in compliance with privacy laws. Audit trails should be protected against tampering.

Privacy

- Privacy concerns are paramount, especially for personal data. Applications must adhere to privacy laws and user preferences, allowing users to specify how their data is handled and ensuring that private information is not exposed.

Encryption and Its Applications

Encryption Basics

- Encryption is the process of transforming data into an unreadable form, which can only be reversed (decrypted) with the appropriate decryption key. Encryption algorithms use encryption keys, and sometimes decryption keys, to secure data.

Historical Use

- Encryption has been used historically to protect messages during transmission. Even if intercepted, unauthorized users couldn't understand the message without the encryption key. Today, it's widely used for securing data during transmission over the internet and mobile networks.

Database Encryption

- Encryption in databases is used to securely store sensitive data. It prevents unauthorized access to data even if a breach occurs. This is crucial for safeguarding personal information like credit card numbers and social security numbers, as data breaches can lead to identity theft.

Encryption Techniques

- Symmetric-Key Encryption: Uses the same key for both encryption and decryption.
- Public-Key (Asymmetric-Key) Encryption: Uses a pair of keys, public and private, for encryption and decryption. Public keys can be freely shared, but private keys must remain secret.

Advanced Encryption Standard (AES)

- A widely adopted symmetric-key encryption algorithm that operates on 128-bit data blocks and can use encryption keys of 128, 192, or 256 bits.

Public-Key Encryption

- Based on mathematical properties, it ensures that even if the public key is known, it's extremely difficult to determine the private key. This method enables secure data exchange and is computationally expensive.

Encryption Support in Databases

- Databases can employ encryption at different levels, including disk-block-level encryption and attribute-level encryption. Key management is crucial, and often a master encryption key is used to protect other keys.

Authentication with Encryption

- Password-based authentication can be enhanced with encryption to prevent eavesdropping during transmission. Challenge-response systems and smart cards are used for secure authentication.

Digital Signatures

- Public-key encryption can be used to create digital signatures, which authenticate data and ensure non-repudiation. This is useful for verifying the authenticity of data.

Digital Certificates

- Certificates are issued by certification authorities to authenticate the public keys of entities. They are used in web browsers to verify the authenticity of websites (HTTPS) and authenticate users.

Class activity

- Application programs
- Web interfaces to databases
- HTML
- Hyperlinks
- Uniform resource locator (URL)
- Forms
- Hypertext Transfer Protocol(HTTP)
- Connectionless protocols
- Cookie
- Session
- Servlets and Servlet sessions
- Server-side scripting
- Java Server Pages (JSP)
- PHP
- Client-side scripting
- JavaScript
- Document Object Model (DOM)
- Ajax
- Progressive Web Apps
- Application architecture
- Presentation layer
- Model-view-controller (MVC) architecture
- Business-logic layer
- Data-access layer
- Object-relational mapping
- Hibernate
- Django
- Web services
- RESTful web services
- Web application frameworks
- Connection pooling
- Query result caching
- Application security
- SQL injection
- Cross-site scripting (XSS)
- Cross-site request forgery (XSRF)
- Authentication
- Two-factor authentication
- Man-in-the-middle attack
- Central authentication
- Single sign-on
- OpenID
- Authorization
- Virtual Private Database (VPD)
- Audit trail
- Encryption
- Symmetric-key encryption
- Public-key encryption
- Dictionary attack
- Challenge–response
- Digital signatures
- Digital certificates

Tools

- IDEs for Web Development:
  - Eclipse: An open-source IDE with extensive plugins for web development.
  - NetBeans: Another open-source IDE with strong support for Java and web development.
  - IntelliJ IDEA: A commercial IDE with free licenses for certain users; popular for Java and web development.
  - Visual Studio: Microsoft's IDE with robust web application development support.
- Application Servers:
  - Apache Tomcat: A popular open-source servlet container for Java web applications.
  - Glassfish: An open-source application server for Java EE (Enterprise Edition).
  - JBoss/WildFly: Red Hat's offerings for Java EE application servers.
  - Resin: A Java EE-compatible application server.
  - Apache Web Server: A widely-used open-source web server.
- Microsoft Technologies:
  - IIS (Internet Information Services): A web and application server for Windows platforms, supporting ASP.NET.
- JavaScript Libraries:
  - jQuery: A widely-used JavaScript library for simplifying HTML DOM traversal and manipulation, as well as event handling.
- Mobile App Development:
  - Android Studio: The official IDE for Android app development.
  - Xcode: Apple's IDE for iOS and macOS app development.
  - AppCode: An IDE by JetBrains for Objective-C and Swift development.
  - Flutter: Google's UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase.
  - React Native: A framework for building mobile apps using React and JavaScript.
- Application Security:
  - OWASP (Open Web Application Security Project): Provides resources and tools for web application security, helping developers secure their applications against common vulnerabilities.

Part five

Big Data Analytics

Chapter 10
big data

Big data refers to extremely large and complex data sets that exceed the capabilities of traditional data processing and management tools. These data sets are characterized by the three Vs: volume, velocity, and variety. Big data poses challenges in terms of storage, processing, analysis, and visualization due to its size and complexity.

Motivation

Introduction of Big Data

- The growth of the World Wide Web during the 1990s and 2000s led to the generation of vast amounts of data, which traditional relational databases were not equipped to handle. This data included web server logs, user interaction data, and various types of user-generated content.

Web Logs

- Web server logs, which recorded user interactions with websites, became a valuable source of information for companies. They contained data about which pages users accessed, user demographics, and even transactional data for e-commerce sites.

Characteristics of Big Data

- Big Data is characterized by its volume, velocity, and variety:
  - Volume: The amount of data is much larger than what traditional databases could manage, often requiring the use of thousands of machines.
  - Velocity: Data arrives at a high rate, necessitating real-time or near-real-time processing.
  - Variety: Data comes in various forms, including semi-structured data, textual data, and graph data, making it different from traditional relational data.

Challenges with Traditional Databases

- Traditional relational databases, while effective for structured data, struggled to handle the scale, speed, and diversity of Big Data. SQL, the standard query language for relational databases, wasn't always suitable for querying diverse data types.

New Data Processing Tools

- As a result, new languages and frameworks were developed to handle Big Data, especially when it required parallel processing on large clusters of machines. These tools allowed for complex queries on diverse data types.
  - Apache Hadoop, Apache Spark, Apache Cassandra, Apache Kafka, Apache Storm, Apache Hive, RapidMiner, Apache Flink, KNIME, Tableau, MongoDB, Apache Mahout, OpenRefine, Sqoop, Pig, Presto, MapReduce, Apache Hbase, Apache Drill, Cloudera, Microsoft Excel, Apache Flume, Qubole, Inc., Elasticsearch

Sources and Uses of Big Data

- The sources of Big Data expanded beyond web logs to include mobile app data, transaction data, sensor data, and metadata from communication networks. Organizations used Big Data for purposes such as user engagement, advertising, business intelligence, and more.

Querying Big Data

- While SQL is widely used for relational databases, querying Big Data requires different approaches. Two main categories of applications were discussed:
  - Transaction-Processing Systems: These require high scalability but may relax some relational database features. Key-value stores are often used.
  - Query Processing Systems: These support non-relational data stored in files and often require parallel processing and the ability to run arbitrary program code.

Processing Textual Data

- Big Data applications often involve processing large volumes of text, image, and video data. Traditional SQL constructs are not well-suited for such tasks, which require parallelization and handling of massive data sizes.

Big Data Storage Systems

Distributed File Systems

- Distributed file systems store files across multiple machines while providing a single-file-system view to clients. They are suitable for storing large files, such as log files, and can also serve as a storage layer for systems that support record storage. Examples include Google File System (GFS) and Hadoop File System (HDFS).

Sharding Across Multiple Databases

- Sharding involves partitioning records across multiple databases or systems. Each database is a traditional centralized database, and client software is responsible for tracking and routing queries to the appropriate database based on partitioned records. It is often used to scale applications with many users.

Key-Value Storage Systems (NoSQL)

- Key-Value storage systems allow records to be stored and retrieved based on a key. While they may provide limited query facilities, they are not full-fledged database systems and are often referred to as NoSQL systems. Examples include MongoDB, Cassandra, and Dynamo.

Parallel and Distributed Databases

- Parallel and distributed databases provide a traditional database interface but store data across multiple machines. They perform query processing in parallel across multiple machines and are suitable for both transaction processing and analytical queries.

Replication and Consistency

- Replication is essential for ensuring data availability and fault tolerance. However, achieving consistency in distributed systems can be challenging, especially in the presence of network partitions. Different systems make trade-offs between availability and consistency.

Scalability

- The choice of storage system often depends on the scalability requirements of the application. Some applications use a combination of storage systems to balance scalability and features like SQL support.

In-Memory Caching

- In-memory caching systems like Memcached or Redis are used to cache data from databases for fast and scalable read-only access. Updates are typically performed on the database, and the application is responsible for updating the cache.

The MapReduce Paradigm

MapReduce Paradigm

- The MapReduce paradigm is a programming model for parallel processing, particularly designed for handling large-scale data processing tasks. It involves breaking down a task into two main functions: the map() function, which processes input data and emits key-value pairs, and the reduce() function, which aggregates and processes those key-value pairs based on their keys. It's commonly used for distributed data processing and is known for its scalability and fault tolerance.

Map Function (map())

- In the context of MapReduce, the map() function is a user-defined function that processes individual input records and produces intermediate key-value pairs. Each input record is passed through the map() function, which can apply some processing to the data and emit zero or more key-value pairs based on that data.

Reduce Function (reduce())

- The reduce() function is another user-defined function in the MapReduce paradigm. It takes a set of intermediate key-value pairs that have the same key (usually generated by the map() function) and performs an aggregation or computation on those values. The output of the reduce() function is typically a set of key-value pairs where each key is unique, and values have been aggregated or processed in some way.

Hadoop

- Hadoop is an open-source framework for distributed storage and processing of large datasets. It provides an implementation of the MapReduce programming model, allowing users to write MapReduce jobs to process data stored in the Hadoop Distributed File System (HDFS). Hadoop is known for its ability to handle big data, fault tolerance, and scalability.

TextInputFormat

- TextInputFormat is a specific input format in Hadoop used for processing text files. It divides the input text file into lines, treating each line as a separate input record for the map() function.

Distributed File System (HDFS)

- HDFS is a distributed file system designed to store and manage large datasets across a cluster of commodity hardware. It's a core component of the Hadoop ecosystem and provides fault tolerance and high-throughput access to data.

SQL on MapReduce

- This concept refers to the use of MapReduce or similar frameworks to process data using SQL-like queries. While traditional relational databases use SQL for querying structured data, SQL-on-MapReduce systems provide a way to execute SQL-like queries on unstructured or semi-structured data stored in distributed file systems.

Hive, SCOPE, ~~Pig~~

- These are examples of systems and languages that allow users to express SQL-like queries or data transformations that can be executed on distributed data using MapReduce or similar frameworks. Hive uses a variant of SQL, SCOPE is a Microsoft system for big data processing, and ~~Pig~~ uses a language called ~~Pig~~ Latin to express data transformations.

Beyond MapReduce: Algebraic Operations

Motivation for Algebraic Operations

- Expressing complex tasks, such as joins, as single algebraic operations can simplify the job of programmers compared to expressing them indirectly through map and reduce functions. Algebraic operations can also be executed efficiently in parallel, making them valuable for processing large datasets.

Extension to Complex Data Types

- While traditional relational algebra deals with data consisting of columns with atomic data types, modern data processing systems like Spark need to handle more complex data types. These systems support operations on datasets containing records with complex data types and return datasets with similar complex data types.

Unifying Framework

- Algebraic operations are treated as a unifying framework for various data operations, including relational operations like joins, machine learning algorithms, and data analytics. These operations can be modeled as algebraic operators that take datasets as inputs and produce datasets as outputs.

Apache Tez and Apache Spark

- Apache Tez provides a low-level API suitable for system implementors, while Apache Spark offers higher-level APIs for application programmers.

Resilient Distributed Dataset (RDD)

- In Apache Spark, data is represented using Resilient Distributed Datasets (RDDs), which are collections of records that can be stored across multiple machines. RDDs are distributed and resilient to failures, ensuring data availability even if some machines fail.

Lazy Evaluation

- Spark uses lazy evaluation, which means that operations on RDDs create a logical plan but do not execute immediately. The entire plan is evaluated only when required by specific actions, allowing for query optimization.

Dataset Type

- In addition to RDDs, Spark also supports the Dataset type, which is suitable for structured data with attributes. Dataset operations are well-suited for relational data and can be used with file formats like Parquet, ORC, and Avro.

Other Algebraic Operations

- Spark supports a wide range of algebraic operations beyond relational ones, including those related to machine learning. These operations can be applied to Datasets, making Spark a versatile tool for various data processing tasks.

Streaming Data

Applications of Streaming Data

- Stock Market: Streaming data is used in stock markets to analyze trades in real-time, identify patterns, and make buying or selling decisions. Regulators also use streaming data to detect illegal trading activities.
- E-commerce: Streaming data is generated from user purchases and searches on e-commerce websites. This data is used for various purposes, such as monitoring the impact of advertising campaigns, detecting sales spikes, and preventing fraudulent activities.
- Sensors: Sensors in vehicles, buildings, and factories continuously send readings as streaming data. These readings are monitored in real-time to detect abnormalities and faults.
- Network Data: Organizations monitor computer networks using streaming data to detect network problems and security threats in real-time. This involves aggregating and processing network data.
- Social Media: Social media platforms receive a continuous stream of messages (posts or tweets) from users. Companies may monitor social media streams for sentiment analysis, user interactions, and campaign impact assessment.

Querying Streaming Data

- Continuous Queries: Analysts can use SQL or relational algebra operations to define continuous queries that run continuously on incoming streaming data. The query results are updated in real-time as new data arrives.
- Stream Query Languages: These are query languages designed for streaming data and often include windowing operations to define specific time intervals or conditions for data processing. Stream query languages separate streaming data from stored relations and allow for efficient real-time processing.
- Algebraic Operators on Streams: This approach allows users to define custom operators (user-defined functions) that operate on each incoming tuple. Operators can maintain internal state and aggregate data as it arrives. This approach offers flexibility but requires custom coding.
- Pattern Matching: Complex Event Processing (CEP) systems allow users to define rules with patterns and actions. When the system detects matching patterns in the stream, it triggers the specified actions.

Graph Databases

Graphs as Data Models

- Graphs are used to model various types of data, such as computer networks (routers and links), road networks (intersections and road links), web pages (hyperlinks), and even enterprise data models (entities and relationships). Graphs provide a flexible way to represent complex relationships between data elements.

Relational Model for Graphs

- While graphs can be represented in relational databases using two relations ("node" and "edge"), this simplistic representation may not be sufficient for complex database schemas. In practice, there can be multiple relations for nodes and edges, each with its own set of attributes.

Graph Databases

- Specialized graph databases like Neo4j offer several advantages over traditional relational databases for handling graph data:
  - They provide a dedicated syntax for defining nodes and edges.
  - They support query languages tailored for expressing path queries and complex graph traversals.
  - They offer efficient implementations for executing graph queries.
  - They may provide additional features like graph visualization.

Cypher Query Language

- Neo4j uses the Cypher query language for expressing graph queries. The example query shown in the passage demonstrates how to match nodes (instructors and students) and traverse edges (advisors) in a graph to retrieve specific information.

Parallel Graph Processing

- For very large graphs, parallel processing is essential. Two common approaches for parallel graph processing are discussed:
  - Map-Reduce and Algebraic Frameworks: These frameworks can be used to represent graph algorithms as computations associated with vertices. However, they may be inefficient for iterative algorithms or long path traversals.
  - Bulk Synchronous Processing (BSP) Frameworks: BSP frameworks, like Pregel and Apache Giraph, focus on iterative processing of graph data. Computation is associated with vertices, and messages are exchanged between vertices in super steps. This approach is more efficient for certain types of graph algorithms.

Graph Processing with Apache Spark

- Apache Spark's GraphX component provides support for graph computations on large graphs. It offers operations like map functions, joins, and aggregations, which can be executed in parallel to handle large-scale graph data.

Class activity

- Volume
- Velocity
- Conversion
- Internet of things
- Distributed file system
- NameNode server
- DataNodes machines
- Sharding
- Partitioning attribute
- Key-value storage system
- Key-value store
- Document stores
- NoSQL systems
- Shard key
- Parallel databases
- Reduce key
- Shuffle step
- Streaming data
- Data-at-rest
- Windows on the streams
- Continuous queries
- Punctuations
- Lambda architecture
- Tumbling window
- Hopping window
- Sliding window
- Session window
- Publish-subscribe systems
- Pub-sub systems
- Discretized streams
- Superstep

Tools

- Distributed File Systems:
  - Apache HDFS (Hadoop Distributed File System): A distributed file system designed to store and manage large volumes of data across clusters of commodity hardware.
- Distributed/Parallel Key-Value Stores:
  - Apache HBase: A distributed, scalable, and consistent NoSQL database that provides real-time read/write access to large datasets.
  - MongoDB: A popular NoSQL database that stores data in flexible, semi-structured BSON format, making it suitable for a wide range of applications.
  - Apache Cassandra: A highly scalable NoSQL database designed for handling large amounts of data across commodity servers.
  - Riak: A distributed NoSQL database that offers high availability, fault tolerance, and scalability.
- Hosted Cloud Storage Systems:
  - Amazon S3 (Simple Storage Service): A scalable object storage service offered by AWS, suitable for storing and retrieving large amounts of data.
  - Google Cloud Storage: Google's object storage service that allows users to store and retrieve data in the cloud.
- Hosted Key-Value Stores:
  - Google BigTable: Google's distributed, scalable NoSQL database for handling large datasets with low latency.
  - Amazon DynamoDB: A managed NoSQL database service by AWS that offers seamless scalability and high availability.
  - Scalable Parallel Databases:
  - Google Spanner: A globally distributed, strongly consistent, and horizontally scalable database system.
  - Cockroach DB: An open-source distributed SQL database designed for scalability, resilience, and strong consistency.
- MapReduce and Data Processing:
  - Apache Hadoop: A framework for distributed storage and processing of large datasets using the MapReduce programming model.
  - Apache Spark: An open-source cluster computing framework for big data processing, providing in-memory data processing capabilities.
  - Apache Tez: A data processing framework that uses a Directed Acyclic Graph (DAG) of data processing tasks.
- SQL Implementations and Query Processing:
  - Apache Hive: A data warehousing and SQL-like query language for big data processing.
  - Apache Impala: A massively parallel processing SQL query engine for Hadoop.
- Stream Processing Systems:
  - Apache Kafka: A distributed streaming platform for building real-time data pipelines and streaming applications.
  - Apache Flink: An open-source stream processing framework for big data analytics.
- Graph Processing Platforms:
  - Neo4J: A popular graph database for managing and querying graph data.
  - Apache Giraph: An open-source graph processing framework based on the Bulk Synchronous Parallel (BSP) model.