SQL & NoSQL Databases

**Slide 1: Title Slide**

* Title: Understanding SQL & NoSQL Databases
* Your Name
* Date

**Slide 2: Introduction**

Definition of Databases

* Structured Collection
* Efficient Storage and Retrieval
* Data Integrity and Security
* Data Relationships:
* Querying and Manipulation

**Brief Mention of SQL and NoSQL**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **SQL Databases** | **NoSQL Databases** |
| Data Structure | Tabular structure with predefined schema | Flexible, dynamic data structure |
| Relational Model | Yes | No |
| Examples | MySQL, PostgreSQL, SQL Server, Oracle | MongoDB, Redis, Cassandra, Neo4j |
| Scalability | Vertical (scaling up) | Horizontal (scaling out) |
| Complex Queries | Well-suited for complex queries | May lack the full power of SQL queries |
| Use Cases | ACID compliance, structured data | Big data, real-time analytics, dynamic data |

**Slide 3: SQL Databases**

**3.1: What is SQL?**

Definition of SQL (Structured Query Language)

* a specialized programming language designed for managing and manipulating relational databases.
* provides a standardized way to interact with databases (querying, inserting, updating, and deleting data)
* primarily designed for relational databases (tables with rows and columns)
* allows users to write structured queries to retrieve, filter, and manipulate data. Queries are composed of specific keywords and clauses like SELECT, FROM, WHERE, JOIN, and GROUP BY.
* supports various operations for manipulating data, including inserting new records, updating existing records, deleting records, and altering the structure of tables.
* provides mechanisms for controlling access to the database, including creating users, granting and revoking privileges, and setting permissions.

Relational Model

* a conceptual framework for organizing and managing data in a database.
* introduced by Dr. E.F. Codd in 1970
* **Tabular Structure**: Data is organized into tables, also known as relations. Each table consists of rows (tuples) and columns (attributes). Rows represent individual records, while columns represent specific pieces of information about those records.
* **Primary Keys**: Each table has a primary key, which is a unique identifier for each row.
* **Foreign Keys**: Relationships between tables are established using foreign keys.
* **Normalization**: The process of minimizing redundancy and dependency by organizing data into separate tables. Helps to prevent data anomalies and ensures data integrity.
* **ACID Properties**: Relational databases adhere to ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure that transactions are processed reliably and consistently.
* **Data Integrity**: The relational model enforces data integrity by allowing constraints to be defined. Constraints, such as uniqueness, referential integrity, and check constraints, help maintain the accuracy and consistency of the data.

**3.2: Types of SQL Databases**

MySQL

* an open-source relational database management system (RDBMS).
* Widely used for web applications and in the LAMP (Linux, Apache, MySQL, PHP) stack
* Known for its speed and performance.
* Commonly used in web development, content management systems, and e-commerce platforms.

PostgreSQL

* open-source, object-relational database management system (ORDBMS).
* support for custom data types, extensibility, and complex queries.
* Supports a wide range of data types and indexing options.
* Often used in enterprise-level applications, data warehousing, and GIS (Geographic Information System) applications.

Microsoft SQL Server

* Microsoft SQL Server is a relational database management system developed by Microsoft.
* Offers a range of editions including Express, Standard, and Enterprise, catering to different needs and budgets.
* Provides seamless integration with other Microsoft products and services.
* Offers robust security features and supports business intelligence and reporting.
* Widely used in enterprises, especially those that rely heavily on other Microsoft technologies like .NET and Windows Server.

Oracle Database

* Oracle Database is a proprietary multi-model database management system.
* Known for its scalability, performance, and robust feature set.
* Offers support for various data models, including relational, object-relational, and JSON.
* Includes advanced features like partitioning, parallel processing, and extensive data management tools.
* Commonly used in large enterprises, particularly for mission-critical applications and data-intensive tasks.

**3.3: Data Modeling in SQL**

* Tables
* Relationships (Primary Key, Foreign Key)
* Normalization

**Slide 4: SQL Queries**

* Basic SQL Syntax
* SELECT, INSERT, UPDATE, DELETE
* Joins
* Aggregations (GROUP BY, COUNT, SUM, etc.)

**Slide 5: ACID Properties**

Explanation of ACID (Atomicity, Consistency, Isolation, Durability)

* ACID compliance is particularly important in applications where data accuracy and reliability are paramount, such as financial systems, healthcare databases, and other mission-critical applications.
* provide a framework for ensuring that transactions are processed in a dependable manner, even in the face of unexpected events or concurrent access by multiple users.

1. **Atomicity**:

* Ensures that a transaction is treated as a single, indivisible unit of work.
* Either completes successfully in its entirety, or it has no effect at all.
* E.g. Consider a bank transfer where money is withdrawn from one account and deposited into another. Atomicity ensures that either both operations (withdrawal and deposit) occur, or neither occurs. There won't be a scenario where only one operation is completed.

1. **Consistency:**

* Ensures that a database transitions from one valid state to another valid state after a successful transaction.
* The database remains in a consistent state before and after the transaction.
* E.g., In an e-commerce application, if a product's quantity is reduced due to a purchase, the database should maintain consistency by ensuring that the product's quantity is never negative.

1. **Isolation:**

* Isolation ensures that multiple transactions can execute concurrently without interfering with each other.
* Each transaction should be isolated from the effects of other transactions until it is committed.
* E.g., Imagine two users making withdrawals from the same account simultaneously. Isolation ensures that each user sees the account balance before the other user's withdrawal is reflected. This prevents conflicts and ensures the transactions do not interfere with each other.

1. **Durability:**

* Durability guarantees that once a transaction is committed, its effects persist even in the event of system failures (e.g., power outage, hardware failure).
* The changes made by the transaction are permanently stored in the database.
* E.g., If a user receives a confirmation message for a successful order in an e-commerce application, the information about that order should be durably stored in the database, even if there is a system crash shortly after.

**How SQL Databases Maintain ACID Properties**

1. **Atomicity**:
   * **Transaction Logs**: SQL databases use transaction logs to record all changes made during a transaction. If a transaction fails, the database can use the log to roll back the changes and restore the data to its previous state.
   * **Rollback Mechanism**: In the event of a failure (e.g., an error or system crash), the database can use the transaction log to undo the effects of an incomplete or failed transaction. This ensures that either all changes are applied or none.
2. **Consistency**:
   * **Constraints and Validation**: SQL databases allow the definition of constraints (e.g., unique constraints, foreign key constraints) and data validation rules. These ensure that data remains consistent with the defined rules, even during transactions.
   * **Data Validation**: Before committing a transaction, the database checks if the changes would violate any constraints or rules. If so, it rejects the transaction, maintaining consistency.
   * **Data Integrity Checks**: The database performs integrity checks to ensure that relationships between tables remain valid, and that data modifications do not lead to inconsistent states.
3. **Isolation**:
   * **Concurrency Control**: SQL databases employ concurrency control mechanisms to manage multiple transactions executing concurrently. Techniques like locking (e.g., read locks, write locks) and isolation levels (e.g., READ COMMITTED, REPEATABLE READ) are used to control access to data.
   * **Transaction Isolation Levels**: Isolation levels define the degree to which transactions are isolated from one another. They control the visibility of changes made by other transactions during the execution of a transaction.
   * **Serializability**: The highest level of isolation (Serializable) ensures that transactions appear to be executed one after the other, even when executed concurrently.
4. **Durability**:
   * **Write-Ahead Logging (WAL)**: SQL databases use WAL to ensure durability. Before a change is committed, it is first written to a log file. Once the change is confirmed in the log, it is applied to the database. In the event of a crash, the database can use the log to recover and apply any uncommitted changes.
   * **Checkpointing**: Periodically, the database creates checkpoints where it flushes modified data from memory to disk. This ensures that changes are permanently stored even if the system crashes.
   * **Redundant Storage**: Some SQL databases may replicate data across multiple servers or have redundant storage systems to ensure that data is not lost in the event of hardware failures.

**Slide 6: NoSQL Databases**

**6.1: What is NoSQL?**

Definition of NoSQL

* SQL databases), which follow a structured, tabular data model, NoSQL databases are designed to handle various types of unstructured, semi-structured, and structured data.
* Well-suited for scenarios where high scalability, flexibility, and performance are critical.

Non-Relational Model

* Type of database management system (DBMS) that does not strictly adhere to the principles of the traditional relational model.
* NoSQL databases offer a more flexible approach to data storage and retrieval.
* Many NoSQL databases are designed to be distributed across multiple nodes or servers. This enables them to scale horizontally, accommodating large volumes of data and high levels of concurrent access.
* NoSQL databases are optimized for specific use cases, such as high-speed data ingestion, real-time analytics, and rapid access to large datasets.

Key Characteristics of NoSQL

1. **Flexible Data Models**: NoSQL databases do not require a fixed schema, allowing for dynamic and flexible data models. They can handle diverse data types and structures, making them suitable for applications with evolving data requirements.
2. **Scalability**: NoSQL databases are designed for horizontal scalability, meaning they can distribute data across multiple servers or nodes. This enables them to handle high volumes of data and many concurrent users.
3. **High Performance**: NoSQL databases are optimized for specific use cases, such as high-speed data ingestion, real-time analytics, and rapid access to large datasets. They can be highly performant in scenarios where immediate data retrieval is crucial.
4. **Designed for Big Data**: NoSQL databases are well-suited for managing and processing big data, which involves the storage and analysis of extremely large datasets that may be too cumbersome for traditional SQL databases to handle efficiently.

**6.2: Types of NoSQL Databases**

Document Stores (MongoDB, CouchDB)

* Store data in flexible, document-like formats (e.g., JSON, BSON). Examples include MongoDB, CouchDB.

Key-Value Stores (Redis, DynamoDB)

* Simple key-value pairs for rapid data access. Examples include Redis, DynamoDB.

Column-Family Stores

* Organize data by columns rather than rows.
* Examples: Cassandra, HBase.

Graph Databases

* Designed for complex relationships in graph structures.
* include Neo4j, Amazon Neptune.

**6.3: Data Modeling in NoSQL**

**Document-Oriented Data Modeling**

* Data is stored in flexible, semi-structured documents (typically in JSON or BSON format), e.g., MongoDB.
* Schema Design: Each document can have different fields, allowing for flexibility. However, it's important to design the schema based on how the data will be accessed.
* For example, in a blog application, a document could represent a blog post, containing fields like title, author, content, and tags.

**Key-Value Data Modeling:**

* Store data as a collection of key-value pairs.
* Values can be simple (strings, numbers) or complex (objects, lists, etc.).
* Redis and DynamoDB
* Schema Design: Each key is unique, and the value can be any type of data. The design often depends on how keys are organized and accessed.
* In a caching system, keys could represent URLs, and the values could be the corresponding HTML content.

**Wide-Column Data Modeling:**

* Designed to handle large amounts of data with high throughput.
* E.g., Apache Cassandra
* Schema Design: Data is organized into rows and columns, like a table in a relational database. However, columns can vary for each row.
* Example: In an IoT system, rows could represent devices, and columns could represent different sensor readings.

**Graph Data Modeling:**

**Slide 7: CAP Theorem**

* Explanation of CAP (Consistency, Availability, Partition Tolerance)
* How NoSQL Databases Align with CAP Theorem

**Slide 8: Scalability and Performance**

* Scalability in NoSQL vs SQL
* Horizontal vs Vertical Scaling

**Slide 9: Use Cases**

**9.1: SQL Use Cases**

* Applications where ACID compliance is crucial.
* Complex queries and reporting

**9.2: NoSQL Use Cases**

* Big Data Applications
* Real-Time Analytics
* Rapid Development and Scaling

**Slide 10: When to Choose SQL vs NoSQL**

**10.1: Factors for Choosing SQL**

* Data Structure is Well-Defined
* Need for ACID Compliance
* Complex Queries and Reporting

**10.2: Factors for Choosing NoSQL**

* Rapid Development and Scaling
* Big Data and Real-Time Analytics
* Flexibility in Data Structure

**Slide 11: Conclusion**

**11.1: Summary**

* Recap of SQL and NoSQL
* Strengths and Weaknesses

**11.2: Choosing the Right Database**

* Consideration of Use Case and Data Requirements

**Slide 12: Q&A**

**Slide 13: References**

Remember to include visuals like diagrams, charts, and examples to illustrate your points. Additionally, be prepared to answer questions during the Q&A session.

Good luck with your presentation! If you have any specific questions or need further assistance on any of the points, feel free to ask.