1. Compare SQL and NoSQL Databases

SQL and NoSQL databases differ fundamentally in structure, scalability, and use cases, each having distinct advantages and disadvantages depending on the requirements.

Data Structure and Schema

- **SQL databases** (Relational Databases) store data in tables with rows and columns and use a fixed schema (predefined structure). This rigid structure enforces data integrity and relationships, making them best for structured data and transactional applications.
- **NoSQL databases** are non-relational and store data in flexible formats such as documents, key-value pairs, graphs, or wide-column stores. They have dynamic or schema-less models, allowing for rapid evolution of data structure and easier handling of unstructured or semi-structured data.

Scalability

- **SQL databases** scale vertically by enhancing the hardware (CPU, RAM) of a single server, which can become costly and have limits.
- **NoSQL databases** excel at horizontal scaling by adding more servers or nodes, making them highly suitable for distributed cloud environments and large-scale data and traffic.

Consistency and Transactions

- **SQL databases** adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring strong consistency and reliable multi-row transactions typical in banking, ERP, or inventory systems.
- **NoSQL databases** often prioritize availability and partition tolerance based on the CAP theorem, with eventual consistency models. Some NoSQL databases offer ACID-like transactions but generally trade off strict consistency for speed and scalability.

Query Language

- SQL databases use the standardized Structured Query Language (SQL).
- NoSQL databases use varied query languages or APIs depending on the data model (e.g., MongoDB uses its own query language for JSON-like documents).

Performance

- SQL databases are optimized for complex queries and relationship-based data.
- NoSQL databases offer better performance for large volumes of unstructured data, high read/write speeds, and real-time processing.

Use Cases

- **SQL**: Best for applications requiring data integrity, complex queries, and structured data such as financial systems, CRM, and ERP.
- **NoSQL**: Ideal for big data, content management, social networks, IoT, and applications needing flexible schemas and horizontal scaling.

Examples

- **SQL**: MySQL, PostgreSQL, Oracle, MS SQL Server.
- **NoSQL**: MongoDB, Cassandra, Redis, CouchDB, Neo4j.

In summary, SQL databases deliver strong consistency and are suited for structured and transactional data, while NoSQL databases provide flexibility, scalability, and speed for handling large-scale, unstructured data in modern applications.

2. Presentation on Normalization vs. Denormalization

Normalization

- A database design technique to reduce data redundancy and inconsistency.
- Organizes data into multiple related tables with minimal duplication.
- Maintains **data integrity** by eliminating anomalies during insert, update, and delete operations.
- Usually increases the number of tables and the complexity of queries due to more joins.
- Optimizes storage space.

Advantages of Normalization

- Reduces data redundancy.
- Ensures data consistency and integrity.
- Efficient use of disk space.

Disadvantages of Normalization

- Complex queries with many joins can degrade read performance.
- Joins can be resource-intensive.

What is Denormalization?

- A technique that intentionally introduces redundancy by combining tables or duplicating data.
- Aims to improve read/query performance by reducing the number of joins.
- Usually reduces the number of tables and simplifies queries.
- Can lead to wasted storage and risk of data inconsistency.
- Often used where read performance is critical, and data updates are less frequent.

Advantages of Denormalization

- Faster query execution and data retrieval.
- Simplified queries.
- Better suited for read-heavy workloads like analytics and reporting.

Disadvantages of Denormalization

- Increased data redundancy and storage costs.
- Risk of data anomalies due to duplicated data.
- More complex and slower write/update operations.

Key Differences

Aspect	**Normalization**	**Denormalization**
Purpose	Reduce redundancy, maintain consistency	Improve read performance by adding redundancy
Data Redundancy	Minimal	Introduced
Number of Tables	Increased	Reduced
Query Complexity	Complex, multiple joins	Simplified, fewer joins
Storage Efficiency	Optimized	More storage required

Aspect	Normalization	Denormalization
Performance	Better for writes	Better for reads
Data Integrity	Maintained	Risk of inconsistencies
Ideal Use Cases	Transactional systems	Read-heavy analytics and reporting

---

Summary

Normalization and Denormalization serve opposite goals in database design. Normalization focuses on eliminating redundant data and ensuring integrity, ideal for systems with frequent writes and updates. Denormalization accepts some redundancy to simplify and speed up read operations, useful in data warehousing, reporting, and scenarios where read performance is crucial.

This understanding helps in designing databases based on specific application needs like consistency vs. performance trade-offs