**Section A: Introduction to SQL/NoSQL**

**1. Choosing SQL or NoSQL for Structured and Semi-Structured Data**

* If the data is highly structured with predefined schemas, **SQL databases (e.g., MySQL, PostgreSQL, SQL Server)** are preferable.
* If the data is semi-structured (JSON, XML) or unstructured (text, images), **NoSQL databases (e.g., MongoDB, Cassandra, Firebase)** are better.

**Example:**

* **SQL:** A banking system storing customer transactions (requires ACID compliance).
* **NoSQL:** A social media app storing user posts and comments (scalability is key).

**2. Challenges in Migrating from SQL to NoSQL**

* **Schema Differences:** NoSQL is schema-less, while SQL has strict schemas.
* **Data Integrity:** NoSQL may not enforce relationships like SQL foreign keys.
* **Query Language:** NoSQL uses different querying methods (e.g., MongoDB’s aggregation pipeline vs. SQL queries).
* **ACID Compliance:** NoSQL databases often use **eventual consistency** instead of strict **ACID compliance**.

**Example:**

* Migrating an e-commerce site from MySQL to MongoDB requires redesigning the schema from relational tables to document-based storage.

**Section B: SQL vs NoSQL**

**3. SQL vs. NoSQL for an E-Commerce Website**

**Advantages of SQL:**

* **Structured Data:** Product catalogs, customer info, and orders follow strict relationships.
* **ACID Compliance:** Ensures transaction reliability.
* **Mature Query Language:** Uses SQL for complex joins and analytics.

**Disadvantages of SQL:**

* **Scalability Issues:** Vertical scaling is costly.
* **Schema Rigidity:** Difficult to modify table structures frequently.

**Advantages of NoSQL:**

* **High Scalability:** Horizontal scaling with sharding.
* **Flexible Schema:** Allows rapid updates.

**Disadvantages of NoSQL:**

* **Weaker Consistency:** Eventual consistency instead of strict ACID compliance.
* **Complex Queries:** No built-in JOINs.

**Best Choice:** Use **SQL for order management and transactions**; use **NoSQL for user-generated content like reviews.**

**4. SQL for Banking Systems**

* Banking systems **must ensure data consistency** (e.g., account balance updates).
* **SQL databases (e.g., PostgreSQL, SQL Server, Oracle)** provide **strong ACID compliance**.
* NoSQL databases prioritize speed but may lead to **inconsistent** transactions.

**Example:**

* If two people withdraw money from the same account, SQL ensures **no data loss** due to **transaction locks**.

**Section C: Managing Databases**

**5. Essential Database Management Tasks**

1. **Backups & Recovery:** Regularly backup data to prevent data loss.
2. **Index Optimization:** Improve query performance by indexing frequently used columns.
3. **Security Management:** Grant and revoke access permissions to prevent unauthorized access.

**6. Optimizing Database Performance for a Streaming Service**

* **Caching:** Store frequently accessed data (e.g., Redis, Memcached).
* **Partitioning:** Split large datasets to improve read/write speeds.
* **Load Balancing:** Distribute database requests across multiple servers.

**Section D: System Databases in SQL Server**

**7. System Databases in SQL Server**

1. **master:** Stores system-level information (logins, server settings).
   * **Use Case:** Recovering system configurations after a crash.
2. **model:** Template for new databases.
   * **Use Case:** Setting default configurations for new databases.
3. **msdb:** Stores job schedules and backup history.
   * **Use Case:** Managing automated backups.
4. **tempdb:** Temporary storage for query processing.
   * **Use Case:** Storing temporary tables and sorting results.

**8. Recovering a Deleted User Database**

* Use **msdb** database to check backup history.
* Restore using:

sql

CopyEdit

RESTORE DATABASE TechShop FROM DISK = 'C:\Backups\TechShop.bak';

**Section E: Normalization**

**9. Converting to 1NF, 2NF, 3NF**

**Unnormalized Table:**

| **OrderID** | **CustomerName** | **Product** | **Quantity** | **SupplierName** | **SupplierContact** |
| --- | --- | --- | --- | --- | --- |
| 101 | John Doe | Laptop | 1 | ABC Ltd. | 1234567890 |
| 102 | Jane Smith | Phone | 2 | XYZ Inc. | 9876543210 |

**First Normal Form (1NF)**

**Issues fixed:**

* **Repeating Groups:** Each field contains atomic values.

| **OrderID** | **CustomerName** | **Product** | **Quantity** | **SupplierName** | **SupplierContact** |
| --- | --- | --- | --- | --- | --- |
| 101 | John Doe | Laptop | 1 | ABC Ltd. | 1234567890 |
| 102 | Jane Smith | Phone | 2 | XYZ Inc. | 9876543210 |

*(Already in 1NF as there are no multiple values in any cell.)*

**Second Normal Form (2NF)**

**Issues fixed:**

* **Partial Dependencies:** Remove non-key attributes from composite primary key.

| **OrderID** | **CustomerID** | | **ProductID** | | **Quantity** |
| --- | --- | --- | --- | --- | --- |
| 101 | 1 | | 201 | | 1 |
| 102 | 2 | | 202 | | 2 |
| **CustomerID** | | **CustomerName** | |
| 1 | | John Doe | |
| 2 | | Jane Smith | |

| **ProductID** | **Product** | | **SupplierID** | |
| --- | --- | --- | --- | --- |
| 201 | Laptop | | 301 | |
| 202 | Phone | | 302 | |
| **SupplierID** | | **SupplierName** | | **SupplierContact** | |
| 301 | | ABC Ltd. | | 1234567890 | |
| 302 | | XYZ Inc. | | 9876543210 | |

**Third Normal Form (3NF)**

**Issues fixed:**

* **Transitive Dependencies:** Remove attributes that don’t directly depend on primary keys.
* SupplierName and SupplierContact were moved to a separate table.
* Now, all attributes only depend on their primary key.

**10. BCNF for Redundancy Reduction**

* **Boyce-Codd Normal Form (BCNF)** removes remaining dependencies.
* **Example:**
  + If a supplier can only supply one product, splitting the table ensures **no redundant supplier details**.

**Final BCNF Tables:**

1. **Orders(OrderID, CustomerID, OrderDate)**
2. **OrderDetails(OrderID, ProductID, Quantity)**
3. **Customers(CustomerID, Name, Contact)**
4. **Products(ProductID, ProductName, SupplierID)**

**Suppliers(SupplierID, SupplierName, Contact)**