**Database Management Assignment:-**

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

1. You are working on a project where you need to store large amounts of structured and semi-structured data. Which type of database (SQL or NoSQL) would you choose and why? Explain with a practical example.

**Solution:**

SQL Databases are best for structured data, ensuring ACID (Atomicity, Consistency, Isolation, Durability) compliance, and maintaining relationships between data like banking system.

But,For storing large amounts of structured and semi-structured data, NoSQL databases are a better choice because they provide scalability, flexibility, and high availability.

Practical Example

A social media platform stores structured data (user profiles, relationships) and semi-structured data (posts, comments, likes). A NoSQL database like MongoDB is ideal because:

* Flexible Schema: User-generated content varies (text, images, videos).
* High Scalability: NoSQL scales horizontally to handle millions of users.
* Fast Read/Write Operations: Ideal for real-time interactions.

1. **A company wants to migrate from a relational database to a NoSQL database for better scalability. What challenges might they face? Discuss with an example**.

**Solution:**

Challenges in Migrating from SQL to NoSQL

1. Data Modeling Differences
   * SQL databases follow a structured schema (tables with fixed columns) whereas NoSQL databases use flexible schemas (document, key-value, column-family, or graph models).
   * Challenge: The existing relational model needs to be redesigned to fit NoSQL’s schema-less structure.
2. Complex Query Handling
   * In NoSQL databases avoid joins for performance reasons but SQL supports JOIN operations to fetch related data.
   * Challenge: Queries need to be rewritten, often leading to data denormalization.
3. Transaction Management
   * SQL databases provide ACID (Atomicity, Consistency, Isolation, Durability) compliance.
   * Many NoSQL databases prioritize eventual consistency for scalability.
   * Challenge: Applications relying on strict consistency may need adjustments.
4. Data Migration Complexity
   * Moving large amounts of structured data to NoSQL requires ETL (Extract, Transform, Load) processes.
   * Challenge: Mapping relational tables to NoSQL documents or key-value pairs.

Example: Migrating a Social Media Platform from SQL to NoSQL

A social media company using PostgreSQL for user profiles, posts, and comments wants to migrate to Cassandra to handle millions of users and scale globally.

Before Migration (SQL - PostgreSQL)

* User data stored in multiple related tables (Users, Posts, Comments).
* Joins used to fetch user posts and comments.
* Strong consistency but struggles with high traffic.

After Migration (NoSQL - Cassandra)

* Data is denormalized and stored in wide-column format.
* No joins, so posts and comments are stored together for fast retrieval.
* High availability and partitioning to handle global users.

Challenges Faced

* Schema redesign: Relational data converted into partitioned key-value format.
* Query changes: SQL queries replaced with Cassandra’s CQL queries.
* Data migration: Large-scale transfer and restructuring required.

**Section B: Advantages and Disadvantages of SQL/NoSQL:**

**3. You are designing an e-commerce website's database. Explain the advantages and disadvantages of using SQL vs. NoSQL in this scenario.**

**Solution:**

* **SQL (Relational Databases):**
  + **Advantages:** SQL databases are best used for structured data ,Strong data integrity, ACID compliance, and faster querying capabilities, making it suitable for complex transactions and data relationships.
  + **Disadvantages:** Can be less flexible for handling unstructured data, and scaling challenges with very high traffic.
* **NoSQL (Non-Relational Databases):**
  + **Advantages:** NoSQL databases are suitable for structured, semi-structured, and unstructured data. As a result, NoSQL databases don't follow a rigid schema but instead have more flexible structures to accommodate their data types and faster query times for specific use cases.
  + **Disadvantages:** Can compromise data consistency and ACID compliance, and complex queries can be challenging.
* **Recommendation:**

For an e-commerce website, a hybrid approach might be used SQL is best for order management, payments, and inventory tracking & NoSQL product catalogs, user activity tracking, and recommendations.

**4. A banking system requires high consistency and ACID compliance. Which database system (SQL or NoSQL) would you recommend? Justify your answer with a real-world use case.**

**Solution:**

* SQL (Relational Databases) is strongly recommended for banking systems due to the critical need for data integrity, consistency, and ACID compliance.
* Banking transactions require ACID properties (Atomicity, Consistency, Isolation, and Durability) to ensure that all transactions are processed correctly and consistently, preventing data corruption or loss.
* Because of the following reasons SQL is suitable for banking system:
  + **ACID Compliance** – Ensures reliable transactions (e.g., money transfer is fully completed or rolled back).
  + **Data Integrity & Security** – Prevents fraud and maintains strict validation.
  + **Complex Queries & Reporting** – Supports financial reports, transaction tracking, and fraud detection.
* Real-world use case: A ₹10,000 transfer from Account A to B, SQL ensures that both debit and credit happen together, avoiding double spending or incorrect balances, which NoSQL’s follows eventual consistency, meaning the balance update might not reflect immediately.This could lead to issues like double spending or incorrect balance calculations.

**Section C: Managing Databases:**

5. **You are a database administrator and need to perform routine maintenance on a production database. Describe at least three essential database management tasks you would perform.**

**Solution:**

**Three essential tasks of database administrator**:

1.Backup and Recovery

* + Regularly create full and incremental backups to prevent data loss in case of failures.
  + Test recovery procedures to ensure quick restoration during emergencies.

2.Performance Optimization

* + Analyze and optimize slow queries using indexing and query tuning.
  + Monitor resource usage (CPU, memory, disk I/O) to prevent bottlenecks.

3.Security & Access Control

* + Regularly update user roles and permissions to prevent unauthorized access.
  + Apply security patches and encrypt sensitive data to protect against breaches.

**6. An online streaming service needs to optimize its database performance. What strategies can be used for effective database management in this case?**

**Solution:**

Strategies to Optimize Database Performance for an Online Streaming Service

1️. Database Caching: Use Memcached to store frequently accessed .Reduces database load and improves response time for users.

2️. Sharding & Replication:

* Sharding: Splits data across multiple servers to distribute traffic efficiently.
* Replication: Creates read replicas to reduce load on the primary database and ensure high availability.

3️. Indexing & Query Optimization: Create indexes on frequently queried fields and Optimize SQL/NoSQL queries to improve search speed and database performance.

**Section D: Identifying System Databases in SQL Server**  
7. List and describe the system databases in SQL Server. Provide one practical use case for each system database.

**Solution:**

SQL Server includes **four main system databases**:

1️. **master**

* Stores **server-wide configuration, logins, linked servers, and system objects**.
* Use Case: Restoring SQL Server settings after a system crash.

2️. **model**

* Serves as a **template for new databases**, inheriting its structure and settings.
* Use Case: Automatically applying predefined configurations to all newly created databases.

3️. **msdb**

* Manages **SQL Agent jobs, backups, alerts, and database mail**.
* Use Case: Scheduling automatic database backups using SQL Server Agent.

4️. **tempdb**

* Stores **temporary tables, stored procedure results, and session data**.
* Use Case: Handling complex queries that require temporary data storage (e.g., sorting large datasets).

**8. You have accidentally deleted a user database in SQL Server. Which system database would you use to recover it, and how?**

**Solution:**

To recover a deleted database, you would use the **msdb** system database because it stores **backup history and job schedules**.

**Recovery Steps:**

**1️. Check Backup History:**

* Use **msdb.dbo.backupset** to verify the latest full backup.

2️.**Restore from Backup:**

* Run RESTORE DATABASE [YourDB] FROM DISK = 'backup\_path.bak' using the latest available backup.

3️.**Recover Transaction Logs:**

* If full recovery mode is enabled, restore the latest transaction log backups to recover recent changes.

**Section E: Normalization Forms (1NF, 2NF, 3NF, BCNF)**  
9. Given the following unnormalized table:

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

Convert it to 1NF, 2NF, and 3NF with proper explanations.

**Solution:**

**First Normal Form (1NF)**

**Rule:** Each column should have atomic values, and there should be no duplicate columns for similar data.

Already in **1NF**, There is no issues in the given table since 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 |

**Second Normal Form (2NF) :**

A table must be in 1NF and all non-key attributes should depend on the whole primary key, not a part of it.

In the given table:

* CustomerName depends only on OrderID, not Product.
* SupplierName and SupplierContact depend only on Product, not OrderID.
* The composite primary key (OrderID, Product) should not have partial dependencies.

So, needed to split into three tables:

**Orders Table :**

|  |  |
| --- | --- |
| **OrderID** | **CustomerName** |
| 101 | John Doe |
| 102 | Jane Smith |

**Order\_Items Table :**

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderID** | **Product** | **Quantity** | **SupplierID** |
| 101 | Laptop | 1 | S1 |
| 102 | Phone | 2 | S2 |

**Suppliers Table :**

|  |  |  |
| --- | --- | --- |
| **SupplierID** | **SupplierName** | **SupplierContact** |
| S1 | ABC Ltd. | 1234567890 |
| S2 | XYZ Inc. | 9876543210 |

**Third Normal Form (3NF)**

A table must be in 2NF, and all non-key attributes should depend only on the primary key (no transitive dependencies).

* SupplierContact depends on SupplierName, not SupplierID.
* No changes needed because SupplierID is already a primary key in **Suppliers Table**, removing transitive dependencies.

1. A company is facing redundancy issues in their database. How would applying BCNF help reduce redundancy? Explain with an example.

**Solution:**

Boyce-Codd Normal Form (BCNF) is an advanced version of 3NF that ensures every determinant is a candidate key. This helps eliminate data redundancy, update anomalies, and insertion anomalies caused by functional dependencies.

For example, if a table contains instructor and department information where one instructor always belongs to a single department, repeating this data across multiple rows leads to redundancy. By decomposing the table into two separate tables (one for courses and another for instructors), we remove redundant data and ensure each table stores only the necessary information.

Thus, BCNF reduces redundancy by ensuring that non-trivial functional dependencies exist only on candidate keys, leading to a more efficient and normalized database design.