**Database Management Assignment**

**Section A: Introduction to SQL**

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.**

Choosing SQL for Storing Large Structured and Semi-Structured Data. SQL (Structured Query Language) is the most widely used language for managing structured and semi-structured data in relational databases. It ensures data integrity, consistency, and efficient querying, making it ideal for applications that require complex transactions and well-organized storage.

* Structured Data Management
* Guarantees Atomicity, Consistency, Isolation, and Durability
* Supports joins, indexing, and aggregation functions to handle large datasets efficiently.
* Semi-Structured Data Support

**Practical Example: E-Commerce Platform**

* Customers (Name, Email, Address)
* Orders (Products, Quantity, Price, Status)
* Payments (Transaction details)

Customers

|  |  |  |  |
| --- | --- | --- | --- |
| CustomerID | Name | Email | Address |
| 1 | Priya | priya@example.com | London |
| 2 | Meera | meera@example.com | New York |

Orders

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID | CustomerID | Product | Quantity | Price |
| 101 | 1 | Laptop | 1 | $1000 |
| 102 | 2 | Phone | 2 | $1400 |

Payments

|  |  |  |  |
| --- | --- | --- | --- |
| PaymentID | OrderID | Amount | Status |
| PAY001 | 101 | $1000 | Completed |
| PAY002 | 102 | $1400 | Pending |

* Ensures smooth order processing & payments
* Prevents duplicate or missing data
* Handles large datasets efficiently with relational queries

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

Challenges in Migrating from a Relational Database to a SQL Database

* Schema Differences
* Data Migration Complexity
* Performance Optimization
* Stored Procedures & Triggers
* Downtime & Business Disruption
* Compatibility Issues

Example: E-Commerce Website Migration (MySQL → MS SQL Server)

A company running an e-commerce platform decides to migrate from MySQL to MS SQL Server for better scalability and performance.

* Data Type Differences – MySQL uses **TINYINT(1)** for boolean values, while MS SQL uses **BIT(1).**
* Stored Procedure Changes – MySQL uses **DELIMITER** for procedures, while MS SQL uses **BEGIN...END.**
* Indexing & Query Optimization – Some queries perform differently due to the change in indexing strategies.

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

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

**Advantages:**

1. Structured Data Management – Organizes data in relational tables for easy retrieval.
2. ACID Compliance – Ensures reliable transactions (e.g., order processing, payments).
3. Efficient Querying & Indexing – Optimized for complex queries using SQL joins and indexing.
4. Data Integrity & Consistency – Maintains accurate and consistent data with constraints.
5. Scalability with Proper Optimization – Can handle large datasets efficiently with indexing and partitioning.
6. Security & Access Control – Provides role-based access, encryption, and authentication for data protection.

**Disadvantages:**

1. Scalability Limitations for High Traffic – Struggles with rapid scaling in high-traffic applications.
2. Rigid Schema Structure – Requires predefined schema, making changes difficult.
3. Complex Joins Can Reduce Performance – Large-scale queries with multiple joins can slow down performance.
4. Higher Setup & Maintenance Costs – Requires dedicated resources for database management.
5. Challenging Horizontal Scaling – Difficult to distribute data across multiple servers compared to NoSQL.

**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.**

**Recommended Database: SQL**

A banking system requires high consistency, accuracy, and transactional integrity, making SQL databases the best choice. SQL databases follow ACID (Atomicity, Consistency, Isolation, Durability) compliance, ensuring reliable financial transactions.

**Justification:**

1. Atomicity – Ensures transactions are fully completed or rolled back (e.g., money transfer fails, no partial deduction).
2. Consistency – Maintains data accuracy across accounts (e.g., balances update correctly after transactions).
3. Isolation – Prevents interference between concurrent transactions (e.g., multiple users withdrawing money simultaneously).
4. Durability – Ensures transaction records persist even after system failures (e.g., transaction history remains intact).

**Real-World Use Case: Online Banking System**

* A customer transfers $500 from Account A to Account B.

The SQL database ensures:

* Deduction from Account A and Addition to Account B happen together.
* If a system failure occurs, the transaction is rolled back, preventing data loss.
* Multiple users making transactions simultaneously do not affect data consistency.

**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.**

As a Database Administrator (DBA), routine maintenance ensures optimal performance, security, and reliability of a production database.

**Tasks:**

1. Database Backup & Recovery – Regularly schedule full and incremental backups to prevent data loss and enable quick recovery in case of failure.
2. Performance Monitoring & Optimization – Analyze query execution, optimize indexing, and remove redundant data to improve database efficiency.
3. Security & Access Control – Manage user roles, permissions, and encryption to protect sensitive data from unauthorized access.

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

**Strategies for Optimizing Database Performance in an Online Streaming Service**

1. Database Indexing – Use indexes on frequently queried columns to speed up searches.
2. Partitioning & Sharding – Distribute large datasets across multiple servers for faster access.
3. Caching Mechanisms – Implement caching to reduce database load.
4. Query Optimization – Analyze slow queries, avoid unnecessary joins, and optimize SQL statements.
5. Load Balancing – Distribute database queries across multiple servers to handle high traffic.
6. Database Connection Pooling – Manage multiple connections efficiently to reduce latency.
7. Regular Data Archiving – Move old or less frequently accessed data to separate storage to improve 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.**

SQL Server includes four main system databases, each serving a specific purpose in database management.

**1. master**

* Stores system-wide metadata, including login accounts, linked servers, and database configurations.
* **Use Case**: If the master database is corrupted, SQL Server cannot start, making it crucial for server recovery.

**2. model**

* Acts as a template for creating new databases, defining default configurations.
* **Use Case:** When a new database is created, it inherits settings (e.g., recovery model) from the model database.

**3. msdb**

* Manages SQL Server Agent jobs, backups, and database maintenance tasks.
* **Use Case:** Used for automating scheduled backups and tracking SQL jobs execution history.

**4. tempdb**

* Stores temporary tables, query results, and system processes. It is recreated on every SQL Server restart.
* **Use Case:** Used for storing temporary result sets in complex queries to improve performance.

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

**System Database Used: msdb**

* The msdb database is a system database in SQL Server that stores important information such as:
  + Backup history
  + SQL Server Agent jobs
  + Restore history
  + Maintenance plans
* Since msdb keeps track of database backups, it plays a crucial role in recovering a deleted database if a backup exists.

**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.**

Step 1: Convert to First Normal Form (1NF)

1NF Rules:

* Ensure each column has atomic (indivisible) values.
* Eliminate repeating groups by creating separate rows for each unique product in an order.

**1NF Table:**

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

Now, all values are atomic, and there are no repeating groups.

Step 2: Convert to Second Normal Form (2NF)

2NF Rules:

* Must be in 1NF.
* Remove partial dependencies (where a non-key attribute depends on part of the primary key).
* Identify the composite primary key (OrderID, Product).

Issue in 1NF:

* CustomerName depends only on OrderID, not on Product.
* SupplierName and SupplierContact depend only on Product, not OrderID.

Solution:

* Split the table into separate entities:
  + Orders Table (stores order details)
  + Products Table (stores product details)

**2NF Tables:**

**Orders Table:**

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

**OrderDetails Table:**

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

**Products Table:**

|  |  |  |
| --- | --- | --- |
| **Product** | **SupplierName** | **SupplierContact** |
| Laptop | ABC Ltd. | 1234567890 |
| Phone | XYZ Inc. | 9876543210 |

Step 3: Convert to Third Normal Form (3NF)

3NF Rules:

* Must be in 2NF.
* Remove transitive dependencies (where a non-key attribute depends on another non-key attribute).

Issue in 2NF:

* SupplierContact depends on SupplierName, not Product.

Solution:

* Create a separate Suppliers Table to remove the dependency.

**3NF Tables:**

**Orders Table:**

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

**OrderDetails Table:**

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

**Products Table:**

|  |  |
| --- | --- |
| **Product** | **SupplierID** |
| Laptop | S1 |
| Phone | S2 |

**Suppliers Table:**

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

Now, all non-key attributes depend only on the primary key, eliminating transitive dependencies.

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

* **BCNF Rule:** A table is in BCNF if it is in 3NF and every determinant is a candidate key.
* Helps eliminate redundancy by resolving anomalies caused by functional dependencies.

**Example of Redundancy Issue in 3NF**

**Unnormalized Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EmployeeID** | **EmployeeName** | **Project** | **ProjectManager** |
| E101 | Rajesh | Project A | Arjun |
| E102 | Priya | Project B | Meera |
| E103 | Suresh | Project A | Arjun |
| E104 | Kavita | Project B | Meera |

**Issue in 3NF:**

* ProjectManager depends on Project, not EmployeeID.
* This causes redundancy, as "Project A" is always managed by "Arjun," but it is repeated multiple times.

**Converting to BCNF**

Step 1: Identify the Functional Dependency Issue

* Primary Key (EmployeeID, Project) → ProjectManager
* Project → ProjectManager (Partial dependency, violating BCNF)

Step 2: Decompose the Table

**Employee\_Project Table (Stores Employee Assignments)**

|  |  |  |
| --- | --- | --- |
| **EmployeeID** | **EmployeeName** | **Project** |
| E101 | Rajesh | Project A |
| E102 | Priya | Project B |
| E103 | Suresh | Project A |
| E104 | Kavita | Project B |

**Project\_Manager Table (Ensures No Redundancy in Managers)**

|  |  |
| --- | --- |
| **Project** | **ProjectManager** |
| Project A | Arjun |
| Project B | Meera |

* ProjectManager is stored only once per project, removing duplicate values.
* Ensures data integrity—if a project manager changes, it only needs to be updated in one place.

Now, the database is free from redundancy and follows BCNF.