**Team 8**

Galvo, Glenn N.  
Dela Cruz, Novrizkie Kent  
Acoymo, Kerlsan S. Jr.  
Escuña, Gilbert P.  
Niñeza, Jason Louis P.

**System Integration and Architecture 1**

1. **Comparison Table: Relational vs NoSQL Databases**

|  |  |  |
| --- | --- | --- |
| Feature | Relational Database (SQL - SQLite3, MySQL, PostgreSQL) | NoSQL Database (MongoDB, Firebase, Cassandra) |
| Structure | Uses tables with predefined schemas | Uses documents, key-value pairs, graphs, or columns |
| Data Integrity | ACID-compliant (ensures data consistency and prevents corruption) | Eventual consistency (data is eventually synchronized but may have temporary inconsistencies) |
| Relationships | Supports JOINs for structured relationships | No JOINs (requires denormalized data, leading to duplication) |
| Query Language | SQL (Structured Query Language) - standard and widely used | Uses NoSQL-specific queries (differs per database type) |
| Scalability | Vertical scaling (better for small to medium applications) | Horizontal scaling (better for large-scale distributed systems) |
| Schema Flexibility | Rigid schema (modifications require migrations) | Flexible schema (easy to change data structure dynamically) |
| Performance | Optimized for structured queries & relational operations | Faster for high-speed transactions and big data |
| Storage Efficiency | Normalized structure (avoids data duplication) | Denormalized structure (data duplication for faster access) |
| Security & Access Control | Strong access controls with user roles and constraints | Security varies (depends on the NoSQL system) |
| Best Suited For | Applications requiring structured relationships, consistency, and complex queries | Applications needing fast, flexible, and large-scale distributed data |

**2. Justification for Choosing Relational Database (SQLite3)**

For a **Library Management System**, a **Relational Database (SQL)** is the **best choice** for the following reasons:

**1. Data Consistency & Integrity (ACID Transactions)**

* Library operations like **borrowing and returning books** require **consistent updates** across multiple tables.
* **Example:** If a user borrows a book:
  + The system must **update the book's status** and **record the transaction**.
  + **ACID transactions in SQL prevent data loss or corruption**.

**2. Structured Data with Defined Relationships**

* **Library systems have structured entities**:
  + **Books, Users, Authors, Borrow Transactions**
* These entities have **clear relationships**, making **foreign keys in SQL ideal**.
* **Example:** A **Books Table** links to an **Authors Table** via a **foreign key constraint**.

**3. Efficient Queries Using JOINs**

* **Library queries involve complex searches**, such as:
  + “Find all overdue books borrowed by a user.”
  + “Show the most borrowed books this month.”
* SQL databases handle such **complex queries efficiently** with **JOINs**, whereas **NoSQL lacks JOIN support**.

**4. Security & Role-Based Access Control (RBAC)**

* **SQL databases provide robust security** through **user authentication and access controls**.
* **Example:**
  + **Librarians** can add/update books.
  + **Users** can only search and borrow books.

**5. Lightweight & Local Storage (for SQLite3)**

* **SQLite3** is **embedded and serverless**, making it:
  + **Easy to set up**
  + **Lightweight** for a small library system
  + **No extra database server needed**

**3. Final Decision: Best Database for Library Management**

|  |  |
| --- | --- |
| ****Criteria**** | ****Best Choice**** |
| ****Database Type**** | ✅ **Relational (SQL - SQLite3)** |
| ****Key Justifications**** | 📌 **Structured relationships, data integrity, efficient queries, and security** |