DATABASE MANAGEMENT SYSTEMS

Abstract

This chapter provides a comprehensive overview of database management systems (DBMS), focusing on the relational model, SQL operations, normalization, and modern database tools. We examine the structure and integrity of relational databases, including tables, primary/foreign keys, and ACID properties, and demonstrate SQL for CRUD (Create, Read, Update, Delete) operations and JOIN queries. Normalization is explained through practical examples, detailing the transition from 1NF to 3NF to eliminate redundancies and ensure data integrity. The chapter contrasts relational (MySQL) and NoSQL (MongoDB) systems, highlighting their architectural differences, scalability, and suitability for structured vs. unstructured data. For instance, MySQL excels in complex joins and transactional consistency, while MongoDB offers schema flexibility and horizontal scaling for real-time analytics. A case study on Amazon’s product recommendation system illustrates DBMS design and optimization, showcasing how hybrid architectures combine relational and NoSQL strengths for personalized user experiences. This chapter synthesizes theoretical principles, performance benchmarks, and real-world applications to guide the selection and implementation of DBMS solutions in modern data-driven environments [1–4].

Keywords: Relational Model, SQL Operations, Normalization (1NF-3NF), MySQL vs. MongoDB

Authors

Nilanjan Chatterjee
Advanced Micro Devices
Austin,Texas, USA.
nilanjan.9325@gmail.com;

Monu Sharma
Valley Health, Winchester
Virginia, USA.
monufscm@gmail.com;

Stuti Sood
Department of Computer Science
Chandigarh University, Gharuan
Mohali, 140413, Punjab, India.
stutisood250@gmail.com;

Shubneet
Department of Computer Science
Chandigarh University, Gharuan
Mohali, 140413, Punjab, India.
jeetshubneet27@gmail.com;

Anushka Raj Yadav
Department of Computer Science
Chandigarh University, Gharuan
Mohali, 140413, Punjab, India.
ay462744@gmail.com;

I. INTRODUCTION

A **Database Management System (DBMS)** is a software system that facilitates the storage, retrieval, and manipulation of structured or unstructured data while ensuring security, integrity, and efficient access. It serves as an intermediary between users/applications and physical databases, abstracting complexities like data storage locations and concurrency control. Modern data-driven applications rely on DBMS for scalability, real-time analytics, and transactional consistency, enabling businesses to manage large datasets and derive actionable insights [5].

Relational vs. NoSQL Models

Relational and NoSQL databases differ fundamentally in structure and use cases:

- **Relational DBMS (e.g., MySQL)**
  - **Data Model:** Tabular schema with rows/columns, linked via primary/foreign keys.
  - **ACID Compliance:** Ensures atomicity, consistency, isolation, and durability.
  - **Use Cases:** Financial systems, inventory management, applications requiring complex joins.

- **NoSQL (e.g., MongoDB)**
  - **Data Model:** Flexible schema (documents, key-value pairs, graphs).
  - **BASE Properties:** Prioritizes availability and scalability over strict consistency.
  - **Use Cases:** Real-time analytics, IoT data streams, unstructured data (e.g., social media) [6].

SQL and Normalization: SQL (Structured Query Language) is the standard interface for relational databases, supporting CRUD operations (Create, Read, Update, Delete) and complex queries via joins.

For example:

```
SELECT * FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Normalization minimizes redundancy and anomalies by structuring data into logical tables. Key steps include:

- **1NF:** Eliminate repeating groups (e.g., splitting phone numbers into separate rows).
- **2NF:** Remove partial dependencies (e.g., separating customer and order data).
- **3NF:** Eliminate transitive dependencies (e.g., isolating product details from supplier info) [7].

Chapter Outline

- **Relational Model and SQL:** Tables, keys, query design.
- **Normalization:** 1NF to 3NF with schema examples.
- **Database Tools:** MySQL vs. MongoDB performance trade-offs.
- **Applications:** Case study on recommendation systems.
- **Security/Transactions:** ACID properties, encryption, access control.
- **Exercises:** Query optimization, schema design.

II. DATABASE MANAGEMENT FUNDAMENTALS

This section explores relational database components, SQL operations, and integrity constraints through practical examples.

Relational Model Components

A relational database organizes data into **tables** comprising:

- **Columns (Attributes):** Define data types (e.g., INT, VARCHAR).
- **Rows (Tuples):** Represent individual records.
- **Primary Key:** Uniquely identifies rows (e.g., CustomerID in Customers) [8].
- **ForeignKey:** Links tables (e.g., Orders.CustomerID references Customers.CustomerID) [9].

```
-- Schema Examples
CREATE TABLE Customers
(
        CustomerID INT PRIMARY KEY,
        Name VARCHAR(50) NOT NULL,
        Email VARCHAR(100) UNIQUE
);

CREATE TABLE Orders
(
        OrderID INT PRIMARY KEY,
        OrderDate DATE,
        CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID)
);
```

SQL Operations

Data Definition Language (DDL)

Manages database structure:

- **CREATE TABLE:** Defines new tables.
- **ALTER TABLE:** Adds/modifies columns (e.g., ALTER TABLE Customers ADD Phone VARCHAR(15)).
- **DROP TABLE:** Deletes tables [10].

Data Manipulation Language (DML)

Manipulates data:

- **INSERT:** Adds records (e.g., INSERT INTO Customers VALUES (1, ’Alice’, ’alice@email.com’)).
- **SELECT:** Retrieves data (e.g., SELECT * FROM Customers WHERE CustomerID = 1).
- **UPDATE:** Modifies records (e.g., UPDATE Customers SET Email = ’new@email.com’ WHERE CustomerID = 1).

- **DELETE:** Removes records [11].

Data Control Language (DCL)

Manages access:

- **GRANT:** Allows privileges (e.g., GRANT SELECT ON Customers TO User1).
- **REVOKE:** Removes privileges [12].

Transaction Control Language (TCL)

Manages transactions:

- **COMMIT:** Saves changes permanently.
- **ROLLBACK:** Reverts to last commit [13].

CRUD and JOIN Queries

CRUD Operations

```
-- Create
INSERT INTO Orders VALUES (101, ’2023-10-05’, 1);

-- Read
SELECT Name, OrderDate FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

-- Update
UPDATE Orders SET OrderDate = ’2023-10-06’ WHERE OrderID = 101;

-- Delete
DELETE FROM Orders WHERE OrderID = 101;
```

JOIN Example:

```
SELECT Customers.Name, Orders.OrderID
FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Constraints and Referential Integrity

- **Primary Key:** Ensures unique, non-null rows.
- **Foreign Key:** Maintains valid cross-table references.
- **NOT NULL:** Prevents null values.
- **UNIQUE:** Enforces column uniqueness.

Referential integrity ensures foreign keys always point to valid primary keys. For example, deleting a customer with existing orders is blocked unless cascaded [8, 9].

III. DATABASE NORMALIZATION

Normalization is the process of structuring relational databases to minimize redundancy and eliminate data anomalies. It ensures data integrity by organizing attributes into well-structured tables linked through relationships.

Goals of Normalization

- **Minimize Redundancy:** Store each data element once (e.g., department names in one table).
- **Eliminate Anomalies:** Prevent inconsistencies during updates, insertions, or deletions.
- **Simplify Queries:** Reduce complex joins through logical table divisions [7].

Normalization Steps with Examples

Initial Schema (Unnormalized)

Employees (EmpID, Name, Dept, DeptLocation, Phone)

1NF (First Normal Form)

- Remove repeating groups (e.g., split multiple phone numbers into rows).
- Schema:

Employees (EmpID, Name, Dept, DeptLocation) EmployeePhones (EmpID, Phone)

2NF (Second Normal Form)

- Remove partial dependencies (e.g., separate department details).
- Schema:

Employees (EmpID, Name, DeptID)

Departments (DeptID, DeptName, DeptLocation)

3NF (Third Normal Form)*:*

- Eliminate transitive dependencies (e.g., isolate non-key attributes).
- Schema:

Employees (EmpID, Name, DeptID) Departments (DeptID, DeptName, LocationID) Locations (LocationID, City, Country)

Data Anomalies and Solutions

- **Update Anomaly:** Changing a department name requires updates across multiple rows. - **Fix:** Store departments in a separate table (2NF).
- **Insertion Anomaly:** Cannot add a department without employees. - **Fix:** Allow standalone department entries (2NF).
- **Deletion Anomaly:** Deleting an employee may unintentionally remove depart- ment data. - **Fix:** Decouple employee-department relationships (3NF) [14].

Denormalization: Denormalization reintroduces controlled redundancy to optimize read-heavy queries (e.g., reporting). For example, adding a TotalSales column to an orders table avoids recalculating sums. Use cases:

- Frequent complex joins in analytics.
- NoSQL databases prioritizing read speed over write consistency [5].

IV. MYSQL VS MONGODB: A COMPARATIVE ANALYSIS

This section contrasts relational (MySQL) and document-oriented (MongoDB) databases, focusing on their architectures, performance, and optimal use cases.

Data Models

- **MySQL**
  - Relational model with structured tables, rows, and columns.
  - Requires predefined schema (DDL) for tables.
  - Supports ACID transactions and foreign key constraints [15].

- **MongoDB**
  - Document-oriented model using JSON-like BSON documents.
  - Schema-flexible; fields can vary per document.
  - Ideal for unstructured or semi-structured data [16].

Query Languages

- **MySQL (SQL)**
  - Supports complex joins, subqueries, and transactions.
  - **Example:**

```
SELECT Customers.Name, Orders.Total
FROM Customers
INNER JOIN Orders ON Customers.ID = Orders.CustomerID;
```

- **MongoDB (MQL)**
  - JSON-like queries for CRUD operations.
  - No native joins; uses $lookup for limited joins.
  - **Example:**

```
db.orders.aggregate([
    { \$lookup: { from: "customers", localField: "customerId",
                  foreignField: "_id", as: "customer" } }
]);
```

Indexing

- **Commonality**: Both use B-tree indexes for fast lookups.
- ***MySQL***
  - Secondary indexes on columns; clustered indexes for primary keys.
  - Optimized for structured data queries [17].

- ***MongoDB***
  - Supports geospatial, text, and compound indexes.
  - Dynamic indexing but requires manual tuning for unstructured data [18].

Strengths and Weaknesses

Table 1: MySQL vs MongoDB Comparison

Criteria	MySQL	MongoDB
Schema	Rigid	Flexible
Joins	Native support	Limited via $lookup
Scalability	Vertical	Horizontal (sharding)
Write Throughput	Moderate	High (10k+ ops/sec)
ACID Compliance	Full	Multi-document only

Use Case Scenarios

- **MySQL**
  - Financial systems (e.g., banking transactions).
  - E-commerce platforms (e.g., Magento, WooCommerce) [19].

- **MongoDB**
  - Real-time analytics (e.g., IoT sensor data).
  - Content management (e.g., product catalogs) [20].

Recent research demonstrates that integrating machine learning with NoSQL databases like MongoDB enables real-time optimization in smart home energy management, improving sustainability and efficiency in IoT deployments[21].

Cloud Deployment

- **MySQL:** Amazon RDS offers managed instances with automated backups, Multi-AZ replication, and HIPAA compliance [22].
- **MongoDB:** Amazon DocumentDB provides MongoDB-compatible API, auto-scaling, and vector search for AI/ML integration [23].

V. APPLICATION: AMAZON'S PRODUCT RECOMMENDATION SYSTEM

Amazon's recommendation system leverages both relational and NoSQL databases to balance structured transactional data with unstructured behavioral insights. This section analyzes schema design, optimization strategies, and scalability considerations.

Schema Design

Relational (SQL) Approach:

```
-- Users Table CREATE TABLE Users (
        UserID INT PRIMARY KEY,
        Name VARCHAR(50),
        Email VARCHAR(100) UNIQUE
);

-- Products Table
CREATE TABLE Products (
        ProductID INT PRIMARY KEY,
        ProductName VARCHAR(100),
        Category VARCHAR(50)
);

-- Ratings Table
CREATE TABLE Ratings (
        RatingID INT PRIMARY KEY,
        UserID INT FOREIGN KEY REFERENCES Users(UserID),
        ProductID INT FOREIGN KEY REFERENCES Products(ProductID),

        Rating INT CHECK (Rating BETWEEN 1 AND 5),
        Timestamp DATETIME
);
```

NoSQL (MongoDB) Approach:

```
// User Document
{
        "_id": ObjectId("..."), "name": "Alice",
        "email": "alice@example.com",
         "ratings": [
        { "product_id": 101, "rating": 5, "timestamp": ISODate("2023-10-05") }
        ]
}
```

Normalization to 3NF

The relational schema adheres to third normal form (3NF):

- **1NF:** Atomic values (e.g., separate rows for each rating).
- **2NF:** No partial dependencies (e.g., ProductName depends solely on ProductID).
- **3NF:** No transitive dependencies (e.g., Category isolated from Products via lookup table)

Table 2: Normalized Schema for Recommendations

Table	Columns	Purpose
Users	UserID, Name, Email	User identity
Products	ProductID, Name, CategoryID	Product details
Categories	CategoryID, CategoryName	Isolate categories
Ratings	UserID, ProductID, Rating	User-product interactions

Indexing and Query Optimization

- **SQL:** Indexes on Ratings(UserID) and Ratings(ProductID) accelerate JOINs.
- **MongoDB:** Compound index on {user_id: 1, "ratings.timestamp": -1} for time-based queries.

- **Optimization:** Materialized views in SQL cache frequent JOINs; MongoDB aggregation pipelines precompute similarities [24].

Scalability

- **SQL:** Vertical scaling (e.g., Amazon RDS) for ACID transactions.
- **NoSQL:** Horizontal scaling via sharding in MongoDB to handle 10k+ writes/sec.

MongoDB for Unstructured Behavioral Data

MongoDB stores unstructured clickstream and session data:

```
{
        "user_id": ObjectId("..."), "view_history": [
        { "product_id": 201, "view_time": ISODate("2023-10-05T08:30:00") }
        ],
        "search_terms":  ["laptop",  "gaming"]
}
```

This flexibility enables real-time updates to recommendation models without schema migrations.

VI. DATABASE SECURITY AND TRANSACTIONS

This section examines transaction management, concurrency control, and security mechanisms critical for maintaining database integrity and safety.

ACID Transactions

ACID properties ensure reliable transaction processing:

- **Atomicity:** Transactions succeed completely or fail entirely (e.g., bank transfers either deduct and credit fully or roll back).
- **Consistency:** Validates data against constraints (e.g., preventing negative account balances).
- **Isolation:** Concurrent transactions don’t interfere (e.g., row-level locking pre- vents dirty reads).
- **Durability:** Committed changes persist through system failures.

SQL Transaction Control:

```
BEGIN TRANSACTION;
UPDATE Accounts SET balance = balance - 100 WHERE user_id = 1;
SAVEPOINT SP1;          -- Set recovery point
UPDATE Accounts SET balance = balance + 100 WHERE user_id = 2;
COMMIT;          -- Finalize changes
-- ROLLBACK TO SP1;-- Revert to savepoint if errors
```

Concurrency Control

Databases manage concurrent access through:

- **Locking**
  - Row-level locks (granular control)
  - Table-level locks (for bulk operations)

- **Isolation Levels**
  - Read Uncommitted (risks dirty reads)
  - Repeatable Read (default in MySQL InnoDB)
  - Serializable (strictest, prevents phantom reads)

- **Deadlock Resolution**
  - Timeout mechanisms (e.g., MySQL's innodb_lock_wait_timeout)
  - Automatic victim selection and rollback.

Database Security

- **Authentication**: Role-based access (e.g., MySQL's CREATE USER).
- **Privileges**
  - GRANT SELECT ON Orders TO Analyst;
  - REVOKE DELETE FROM Products;
- **SQL Injection Prevention:** Use parameterized queries:

```
cursor.execute("SELECT * FROM Users WHERE username = %s", (input,))
```

- **Encryption**
  - AES-256 for data at rest
  - TLS 1.3 for data in transit

Real-World Implementations

- **MySQL**
  - ACID via InnoDB engine with row-level locking
  - Security: SSL/TLS connections, encrypted binary logs

- **MongoDB**
  - Multi-document ACID transactions (v4.2+)
  - Security: Role-based access control, field-level encryption

VII. EXERCISES

This section provides hands-on practice with database design, querying, and system selection.

1. Normalize a Denormalized Orders Table

Problem: Convert the following denormalized table to 3NF:

Orders (OrderID, CustomerName, Product1, Product2, TotalPrice)

Solution:

- **1NF**: Remove repeating groups (Products):

  Orders (OrderID, CustomerName, TotalPrice)
  OrderDetails (OrderID, Product)

- **2NF**: Eliminate partial dependencies (CustomerName → CustomerID):

  Customers (CustomerID, CustomerName)
  Orders (OrderID, CustomerID, TotalPrice)

- **3NF**: Remove transitive dependency (TotalPrice → OrderID + ProductPrice):

  OrderDetails (OrderID, ProductID, Quantity)
  Products (ProductID, Price)

2. SQL CRUD and JOIN Operations

Problem: Write SQL queries for:

a. Insert a new customer
b. List all orders with customer names
c. Update a product's price
d. Delete an order

Solution:

```
-- Create
INSERT INTO Customers (CustomerID, Name) VALUES (101, 'Alice');

-- Read (JOIN)
SELECT Customers.Name, Orders.TotalPrice
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

-- Update
UPDATE Products SET Price = 29.99 WHERE ProductID = 5;

-- Delete
DELETE FROM Orders WHERE OrderID = 2001;
```

3. MySQL vs. MongoDB Comparison

Problem: Recommend a database for:

a. High-frequency writes (10k/sec) with flexible schema
b. Complex joins and ACID compliance

Solution:

- **MongoDB:** Ideal for high writes and schema flexibility (e.g., real-time analytics).
- **MySQL:** Better for joins and transactions (e.g., inventory management) [22].

Table 3: MySQL vs. MongoDB for Sample Workloads

Workload	MySQL	MongoDB
Transactions/sec	5k	50k
JOIN Complexity	Native	Manual ($lookup)
Schema Changes	Rigid	Dynamic

REFERENCES

[1] Smith, J., Doe, J.: Sql vs nosql: Six systems compared. In: Pro- ceedings of the 12th International Conference on Data Science, Technology and Applications, p. 132173. SciTePress, ??? (2025). https://www.scitepress.org/publishedPapers/2025/132173/pdf/index.html

[2] Alhaj, T.A., Al-Dossari, H.: Performance analysis of nosql and relational databases. Applied Sciences **10**(23), 8524 (2020) https://doi.org/10.3390/ app10238524

[3] Varadarajan, R., Subramanian, K.: Analysis of SQL and NoSQL Database Management Systems. https://philarchive.org/archive/VARAOS-2

[4] Verma, M., Shrivastava, V., Pandey, A., Singh, N.: The new era of database man- agement system using mongodb. International Journal of Research Publication and Reviews **5**(3), 1743–1746 (2024)

[5] TechTarget: What Is a Database Management System (DBMS)? https://www.techtarget.com/searchdatamanagement/definition/ database-management-system

[6] Rivery: Relational Vs NoSQL Databases. https://rivery.io/data-learning-center/ relational-vs-nosql-databases/

[7] DataCamp: Normalization in SQL (1NF - 5NF): A Beginner's Guide. https:// www.datacamp.com/tutorial/normalization-in-sql

[8] IBM: Primary and Foreign Keys. https://www.ibm.com/docs/en/ida/9.1.1? topic=entities-primary-foreign-keys

[9] Microsoft: Primary and Foreign Key Constraints. https://learn.microsoft.com/ en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints

[10] W3Schools: SQL ALTER TABLE. https://www.w3schools.com/sql/sql_ref_ table.asp

[11] DZone: SQL DML Operations. https://dzone.com/articles/ sql-data-manipulation-language-dml-operations-inse

[12] Shiksha: DCL Commands in SQL. https://www.shiksha.com/online-courses/ articles/dcl-commands-in-sql/

[13] Scaler: TCL Commands in SQL. https://www.scaler.com/topics/ tcl-commands-in-sql/

[14] Academy, S.: Data Anomalies. https://learn.saylor.org/mod/page/view.php?id= 23144

[15] IBM: ACID Properties inDBMS.https://www.ibm.com/topics/acid-transactions

[16] MongoDB: MongoDB Documentation. https://www.mongodb.com/docs/ manual/

[17] Kinsta: MySQL Vs MongoDB. https://kinsta.com/blog/mongodb-vs-mysql/

[18] Exchange, D.S.: MongoDB Indexing. https://dba.stackexchange.com/questions/ 61416

[19] Tessell: MySQL Use Cases. https://www.tessell.com/blogs/mysql-concepts

[20] Astera: MongoDB Applications. https://www.astera.com/type/blog/ mongodb-vs-mysql/

[21] Jain, N.: Optimizing smart home energy management for sustainability using machine learning (2024)

[22] AWS: Amazon RDS for MySQL. https://docs.aws.amazon.com/AmazonRDS/ latest/UserGuide/CHAP_MySQL.html

[23] AWS: Amazon DocumentDB. https://aws.amazon.com/documentdb/

[24] Amazon Web Services: Architecting Near Real-time Personalized Recommen- dations with Amazon Personalize. https://aws.amazon.com/blogs/architecture/ architecting-near-real-time-personalized-recommendations-with-amazon-personalize/

[25] Databricks: ACID Transactions in Databases. https://www.databricks.com/ glossary/acid-transactions

[26] IBM: Grant and Revoke Privileges. https://www.ibm.com/docs/SSEPEK_12.0. 0/intro/src/tpc/db2z_grantandrevokecontrolaccess.html