Gisma University of Applied Sciences

Assessment Submission Form

Student Number (If this is group work, please include the student numbers of all group participants)	1. Pavan Kakadiya GH1039356 2. Nikhil Ratilalbhai Sojitra GH1039510
Assessment Title	Restaurant Management Database
Module Code	M605
Module Title	Advanced Database
Module Tutor	Dr. Mazhar Hammed
Date Submitted	27 March 2025

Declaration of Authorship

I declare that all material in this assessment is my own work except where there is clear acknowledgement and appropriate reference to the work of others.

I fully understand that the unacknowledged inclusion of another person's writings or ideas or works in this work may be considered plagiarism and that, should a formal investigation process confirms the allegation, I would be subject to the penalties associated with plagiarism, as per GISMA Business School, University of Applied Sciences' regulations for academic misconduct.

Signed…Pavan Kakadiya, Nikhil Ratilalbhai Sojitra . Date ……27 March 2025…

GitHub Link : https://github.com/nikhilsojitra/db
VideoLink : https://drive.google.com/file/d/1_C0pZsnW1LTkRHUgG68rSlYkZArehA1K/view?

INDEX

Introduction

Databases are a crucial tool for storing, retrieving, and modifying data in the field of data management. Efficient database systems are now essential for guaranteeing that data can be accessed and examined in real-time as companies and organizations gather enormous volumes of data. Our project focuses on a hybrid database system that combines NoSQL (Not Only SQL) and SQL (Structured Query Language) databases.

There are different types of NoSQL databases, including:

Document-Based Databases: These store data as documents, usually in JSON or BSON format. Each document can have a unique structure, making it easy to store complex, nested data. Examples include MongoDB and CouchDB.

Key-Value Stores: These are the simplest form of NoSQL databases, where data is stored as key-value pairs. They are ideal for storing session data, caching, or configurations. Redis and DynamoDB are examples of key-value stores.

Column-Family Stores: These databases store data in columns instead of rows, making them highly efficient for read-heavy applications. Examples include Cassandra and HBase.

Graph Databases: These databases are designed to represent and query data in the form of nodes and edges, making them ideal for handling complex relationships like social networks or recommendation systems. Neo4j is a popular graph database.

NoSQL databases are frequently utilized in situations where performance, scalability, and flexibility are essential. For example, they work well with applications like e-commerce platforms, content management systems, and real-time analytics that need quick development cycles or handle massive amounts of unstructured or semi-structured data.

The Role of SQL and NoSQL in the Restaurant Management Databse

In this project, a hybrid approach involving both SQL and NoSQL databases was chosen to handle different types of data efficiently. The SQL database is used for transactional data such as orders, waiters, tables, and dishes. This type of data requires consistency and adherence to a defined schema, making it well-suited for a relational database. The SQL system ensures that data is accurately tracked, such as maintaining relationships between waiters and orders, ensuring that the correct prices are applied to orders, and calculating total costs.

SQL Database Overview

Basic Principles of SQL Databases

SQL databases are built on the relational model, where data is organized into tables. A table in a relational database is a collection of rows and columns, where each row represents a record, and each column represents a field (or attribute) of that record. This tabular format allows for easy manipulation and retrieval of data through queries.

Key Features of SQL Databases

1. **Normalization:** SQL databases rely heavily on the concept of normalization, which is the process of organizing data to reduce redundancy and improve data integrity. Normalization ensures that data is logically stored in different tables to avoid duplication and maintains consistency across the database.
    a) For example, in a restaurant management system, instead of storing the same dish information repeatedly for each order, dishes can be stored in a separate table, and the orders table would reference the dish table through foreign keys.

2. **Data Integrity and Constraints:** SQL databases enforce data integrity by applying constraints. Some common types of constraints include:
    a) NOT NULL: Ensures that a column must have a value and cannot be left empty.
    b) UNIQUE: Guarantees that all values in a column are distinct.
    c) CHECK: Defines conditions that data must satisfy before being inserted or updated (e.g., price must be greater than 0).
    d) FOREIGN KEY: Ensures referential integrity by establishing relationships between tables, ensuring that the foreign key values exist in the related primary key table.

3. **Transactions:** SQL databases support transactions, which are collections of operations that are executed as a single unit. If any part of the transaction fails, the database ensures that all changes are rolled back, maintaining data consistency. Transactions adhere to the ACID properties:
    a) Atomicity: All operations in a transaction are treated as a single unit; they either all succeed or all fail.
    b) Consistency: The database must remain in a valid state before and after the transaction.
    c) Isolation: Transactions are isolated from each other, meaning that intermediate results are not visible to other transactions.
    d) Durability: Once a transaction is committed, its changes are permanent, even in the event of a system failure.

4. **Querying with SQL:** SQL databases allow for powerful querying capabilities using SQL statements. Some key SQL operations include:
    a) SELECT: Retrieves data from one or more tables.
    b) INSERT: Adds new records into a table.
    c) UPDATE: Modifies existing records.
    d) DELETE: Removes records from a table.
    e) JOIN: Combines data from multiple tables based on relationships.
    f) AGGREGATE functions: Such as SUM, COUNT, AVG, MAX, and MIN, which allow for summarizing and analyzing data.

SQL queries allow for complex data manipulation and retrieval, making it a powerful tool for business applications.

Advantages of SQL Databases

- Data Integrity and Consistency: SQL databases are designed to ensure data consistency, making them suitable for applications where accurate data is critical. The use of primary keys, foreign keys, and constraints ensures that the data is valid, accurate, and linked correctly.

Disadvantages of SQL Databases

- Scalability Issues: While SQL databases can scale vertically (by upgrading hardware), they face challenges when it comes to scaling horizontally (distributing data across multiple machines). As applications grow and data volume increases, it may become harder to scale a traditional SQL database.

SQL Database Use Case in Project

In the restaurant management system, an SQL database plays a crucial role in handling structured transactional data, including the management of orders, tables, waiters, and dishes. The relational structure of SQL ensures that these entities can be efficiently related to each other. For example, each order is linked to a waiter and can contain multiple order items (dishes), which are stored in their respective tables. The use of foreign keys ensures the integrity of these relationships.

1. Waiters Table

This table stores information about the waiters working in the restaurant. It includes details such as the waiter's ID, username, email, password, and timestamps for creation and updates.

SQL Code:

```
CREATE TABLE waiters (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Explanation:

- id: The primary key that uniquely identifies each waiter.
- username: The waiter's username for login.
- email: The waiter's email address, which must be unique.

- password: The password for the waiter.
- created_at: Timestamp for when the waiter record was created.
- updated_at: Timestamp for when the waiter record was last updated.

2. Dish Table

This table contains information about the dishes available in the restaurant. Each dish has a unique ID, a name, and a price.

SQL Code:

```
CREATE TABLE dish (
    dish_id INT AUTO_INCREMENT PRIMARY KEY,
    dish_name VARCHAR(255),
    price INT
);
```

Explanation:
- dish_id: The primary key uniquely identifying each dish.
- dish_name: The name of the dish (e.g., "Pizza", "Pasta").
- price: The price of the dish.

3. Orders Table

This table holds the orders made by the waiters for customers. It includes the order ID and the waiter ID that references the waiters table to know which waiter created the order.

SQL Code:

```
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    waiter_id INT,
    FOREIGN KEY (waiter_id) REFERENCES waiters(id)
);
```

Explanation:
- order_id: The primary key uniquely identifying each order.
- waiter_id: A foreign key referencing the id in the waiters table, linking the order to the waiter who created it.

4. Order_items Table

This table contains information about the individual items ordered within an order. It links the orders table and the dish table, allowing the system to know which dishes were ordered and in what quantity.

SQL Code:

```
CREATE TABLE order_items (
```

```
    item_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT REFERENCES orders(order_id),
    dish_name VARCHAR(255),
    quantity INT
);
```

Explanation:

- item_id: The primary key uniquely identifying each item in the order.
- order_id: A foreign key referencing the order_id in the orders table, identifying which order the item belongs to.
- dish_name: The name of the dish ordered (e.g., "Pizza").
- quantity: The quantity of the dish ordered.

CURD Operation:

```
-- Inserting Data
INSERT INTO waiters (username, email, password)
VALUES ('john_doe', 'john.doe@example.com', 'password123');

INSERT INTO dish (dish_name, price)
VALUES ('Pizza', 12), ('Pasta', 8), ('Burger', 10);

INSERT INTO orders (waiter_id)
VALUES (1), (2);

INSERT INTO order_items (order_id, dish_name, quantity)
VALUES (1, 'Pizza', 2), (1, 'Pasta', 1), (2, 'Burger', 1);

INSERT INTO tables (TableNumber, Capacity)
VALUES (1, 4), (2, 2), (3, 6);

-- Updating Data
UPDATE waiters
SET username = 'john_doe_updated'
WHERE id = 1;

UPDATE dish
SET price = 15
WHERE dish_name = 'Pizza';

UPDATE orders
SET waiter_id = 3
WHERE order_id = 1;

UPDATE order_items
SET quantity = 3
WHERE order_id = 1 AND dish_name = 'Pizza';

UPDATE tables
SET Status = 1
```

```
WHERE TableNumber = 1;

-- Reading Data (SELECT queries)
SELECT * FROM waiters;

SELECT * FROM dish WHERE price > 10;

SELECT * FROM orders WHERE waiter_id = 3;

SELECT * FROM order_items WHERE order_id = 1;

SELECT * FROM tables WHERE Status = 1;

-- Deleting Data
DELETE FROM waiters WHERE id = 1;

DELETE FROM dish WHERE dish_name = 'Pasta';

DELETE FROM orders WHERE order_id = 2;

DELETE FROM order_items WHERE order_id = 2;

DELETE FROM tables WHERE TableNumber = 3;

-- Aggregation and Analysis

-- 1. Calculate total sales by dish
SELECT dish_name, SUM(quantity) AS total_quantity_sold, SUM(quantity * price) AS total_sales
FROM order_items oi
JOIN dish d ON oi.dish_name = d.dish_name
GROUP BY dish_name
ORDER BY total_sales DESC;

-- 2. Average rating for each dish from customer reviews
SELECT dish_name, AVG(rating) AS avg_rating
FROM customer_reviews
GROUP BY dish_name;

-- 3. Total revenue generated for each dish from customer orders
SELECT oi.dish_name, SUM(co.total_amount) AS total_revenue
FROM order_items oi
JOIN customer_orders co ON oi.order_id = co.order_id
GROUP BY oi.dish_name;

-- 4. Most ordered dish
SELECT dish_name, COUNT(*) AS order_count
FROM order_items
GROUP BY dish_name
ORDER BY order_count DESC
LIMIT 1;

-- 5. Calculate the total amount of revenue from all orders
SELECT SUM(total_amount) AS total_revenue
```

FROM customer_orders;

Table

This table stores information about the restaurant's tables. It contains the table number, capacity, and status to track whether the table is available for new orders or not.

SQL Code:

```
CREATE TABLE tables (
    id INT PRIMARY KEY AUTO_INCREMENT,
    TableNumber INT NOT NULL,
    Capacity INT NOT NULL,
    Status BOOLEAN NOT NULL DEFAULT 0
);
```

Explanation:

- id: The primary key uniquely identifying each table.
- TableNumber: The number assigned to each table (e.g., Table 1, Table 2).
- Capacity: The number of seats available at the table.
- Status: A boolean indicating whether the table is available for new orders (0 for available, 1 for occupied).

NoSQL Database Overview

Semi-structured or unstructured data can be handled by NoSQL (Not Only SQL) databases. NoSQL databases offer greater flexibility in data representation and storage than SQL databases, which employ structured query languages and tables with rigid relationships between them. They are typically chosen when working with large amounts of data moving quickly or when the data structure changes often.

There are several types of NoSQL databases:

- Document-based (e.g., MongoDB, CouchDB)
- Key-Value Stores (e.g., Redis, DynamoDB)
- Column-family Stores (e.g., Cassandra, HBase)
- Graph Databases (e.g., Neo4j)

Basic Principles of NoSQL Databases

- Schema-less Structure: NoSQL databases do not require a predefined schema, allowing for flexible data models. This is especially useful when dealing with semi-structured data or when the data structure needs to evolve over time.
- Horizontal Scaling: Unlike SQL databases, which scale vertically (upgrading hardware), NoSQL databases are designed for horizontal scaling, meaning they can distribute data across multiple machines seamlessly.

Key Features of NoSQL Databases

1. Flexibility: NoSQL databases provide the flexibility to store different types of data with varying structures. This is especially helpful in dynamic applications where new fields might be added frequently, like customer orders, which can have different attributes depending on the product or service.
2. Scalability: NoSQL databases can scale horizontally, meaning they can handle large amounts of data and high traffic by adding more servers. This makes them an excellent choice for systems that need to handle high-volume data, such as real-time customer interact

Advantages of NoSQL Databases

- High Performance: NoSQL databases can handle massive volumes of data with high velocity. The absence of complex joins in the data structure allows for faster data retrieval.
- Scalability: They are designed to scale out across multiple machines, ensuring the database can grow as your data volume increases.

Ideal for Unstructured Data: They are perfect for dealing with unstructured data such as social media content, videos, and sensor data.

Disadvantages of NoSQL Databases

- Data Consistency: NoSQL databases generally use eventual consistency, meaning data might not be instantly synchronized across all nodes. This can be a challenge in systems requiring real-time consistency.

- Complex Queries: While NoSQL databases are excellent for key-value lookups and document-based storage, complex querying (like JOINs in SQL) can be more challenging and might require additional logic or processing.

NoSQL Database Use Case in Project

For rrestaurant management Databse, NoSQL databases can store non-relational data, such as logs, customer interactions, or analytics data. For example, customer reviews, real-time interactions, or transactions might not fit well into the rigid structure of a SQL database but can be efficiently stored and retrieved in a NoSQL database..

MongoDB Code Structure:

```
// MongoDB Playground
// Use Ctrl+Space inside a snippet or a string literal to trigger completions.

// The current database to use.
use("restaurant_management");

db.createCollection("customer_reviews")

db.createCollection("customer_orders")

//insert data

db.customer_reviews.insertMany([
   {
     "customer_id": 12345,
     "dish_name": "Pizza",
     "rating": 4,
     "comment": "Great taste, but could use more cheese!",
     "timestamp": new ISODate("2025-03-27T14:00:00Z")
   },
   {
     "customer_id": 67890,
     "dish_name": "Burger",
     "rating": 5,
     "comment": "Absolutely delicious!",
     "timestamp": new ISODate("2025-03-26T15:30:00Z")
   }
 ]);


 db.customer_orders.insertMany([
   {
     "order_id": 1,
     "waiter_id": 101,
     "table_number": 5,
     "items": [
       { "dish_name": "Pizza", "quantity": 2, "price": 12 },
       { "dish_name": "Pasta", "quantity": 1, "price": 8 }
```

```
    ],
    "total_amount": 32,
    "timestamp": new ISODate("2025-03-27T12:30:00Z")
  },
  {
    "order_id": 2,
    "waiter_id": 102,
    "table_number": 3,
    "items": [
      { "dish_name": "Burger", "quantity": 1, "price": 10 },
      { "dish_name": "Fries", "quantity": 1, "price": 5 }
    ],
    "total_amount": 15,
    "timestamp": new ISODate("2025-03-26T13:45:00Z")
  }
]);


//retrive data
db.customer_reviews.find({ "dish_name": "Pizza" });

db.customer_orders.find({ "table_number": 5 });

db.customer_orders.find({ "total_amount": { $gt: 30 } });

db.customer_reviews.find().sort({ "timestamp": -1 }).limit(5);

//update data
db.customer_reviews.updateOne(
  { "customer_id": 12345, "dish_name": "Pizza" },
  { $set: { "rating": 5, "comment": "Now it's perfect!" } }
);

db.customer_orders.updateOne(
  { "order_id": 1 },
  { $push: { "items": { "dish_name": "Salad", "quantity": 1, "price": 6 } } }
);


//delete data
db.customer_reviews.deleteOne({ "customer_id": 12345, "dish_name": "Pizza" });

db.customer_orders.deleteMany({ "table_number": 5 });


//Aggregation
db.customer_reviews.aggregate([
  { $match: { "dish_name": "Pizza" } },
  { $group: { "_id": "$dish_name", "average_rating": { $avg: "$rating" } } }
]);

db.customer_orders.aggregate([
  { $unwind: "$items" },
```

```
  { $group: { "_id": "$items.dish_name", "total_sales": { $sum: "$items.quantity" } } },
  { $sort: { "total_sales": -1 } }
]);

db.customer_orders.aggregate([
  { $unwind: "$items" },
  { $group: { "_id": null, "total_revenue": { $sum: "$total_amount" } } }
]);

db.customer_orders.aggregate([
  { $unwind: "$items" },
  { $group: { "_id": "$items.dish_name", "total_orders": { $sum: "$items.quantity" } } },
  { $sort: { "total_orders": -1 } },
  { $limit: 1 }
]);
```

Description of the MongoDB code:

1. Create Collections:
   - db.createCollection("customer_reviews"): Creates a collection for storing customer reviews.
   - db.createCollection("customer_orders"): Creates a collection for storing customer orders.

2. Insert Data:
   - Customer Reviews:
   - Inserts two reviews for dishes (Pizza and Burger), each with a customer ID, rating, comment, and timestamp.
   - Customer Orders:
     - Inserts two customer orders, each with an order ID, waiter ID, table number, items ordered (dish name, quantity, price), total amount, and timestamp.

3. Retrieve Data:
   - Customer Reviews:
     - Retrieves reviews where the dish is "Pizza".
   - Customer Orders:
     - Retrieves orders for table number 5.
     - Retrieves orders where the total amount is greater than 30.
     - Retrieves the latest 5 reviews sorted by timestamp.

4. Update Data:
   - Customer Reviews:
     - Updates a specific review for "Pizza" by customer ID 12345, changing the rating to 5 and the comment to "Now it's perfect!".
   - Customer Orders:
     - Updates the order with order ID 1, adding a new dish ("Salad") to the items array.

5. Delete Data:
   - Customer Reviews:
     - Deletes a specific review for "Pizza" by customer ID 12345.
   - Customer Orders:
     - Deletes all orders associated with table number 5.

6. Aggregation Queries:
   - Customer Reviews:
     - Aggregates the reviews for "Pizza", calculating the average rating.
   - Customer Orders:
     - Total sales per dish: Aggregates the quantity of each dish sold by unwinding the "items" array.

Total revenue: Aggregates the total amount from all orders.
Most popular dish: Aggregates the total quantity sold for each dish and sorts to find the most popular dish.
Top-selling dish: Finds the top-selling dish by the quantity sold, limiting the result to one dish

Conclusion

The **Restaurant Management Database** project successfully integrates SQL and NoSQL databases to optimize restaurant operations. SQL databases manage structured data like orders, waiters, and dishes, ensuring data consistency, integrity, and smooth relationships between different entities. MongoDB, on the other hand, handles unstructured data, such as customer reviews, providing flexibility and scalability.'

The system provides essential features such as table selection, order management, and billing, offering a seamless user experience for both staff and customers. It is a reliable, efficient, and scalable solution that can adapt to future growth, improving restaurant operations and customer satisfaction. This project exemplifies how the right combination of SQL and NoSQL databases can create an effective management tool for the restaurant industry.