Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/40/20230626

Design of management database for Mihiyo company

Zihao Zhong

Computer and Information Security Management Department, Fujian Police College - Fuzhou, China

fwadsworthra85287@student.napavalley.edu

Abstract. The mihoyo Sales Management System was developed to adjust to the dynamic business environment and maintain a competitive edge. In the context of this era, the mihoyo Sales Management System utilizes a robust database structure to optimize sales operations. In this paper, a management database for Mihiyo company is designed. The customer module stores customer information, linked to orders, payments, and expenses. An order table tracks order details, linked to customers and products. Payment and shipment tables manage transaction and delivery information. The inventory table enables real-time monitoring of stock levels. Financial management tables record sales, payment, and expense data. User tables store information related to different user roles. The system's interface seamlessly interacts with the database, allowing users to access and update information efficiently. The reporting and analytics module analyzes data from the database, facilitating decision-making and performance evaluation. Database testing requires completing relevant test cases and achieving a robust system.

Keywords: Company, Data-table, Customer, Product.

1. Introduction

The mihoyo Sales Management System was designed to adapt to the rapidly changing business landscape and stay competitive [1]. Manual processes were inefficient, prompting the need for automation. The system streamlined sales operations, enhanced customer satisfaction, and enabled data-driven decision-making. It centralized customer information, expedited order processing, and provided real-time inventory monitoring [2]. Financial management capabilities ensured accurate tracking for analysis and informed decision-making.

One of the core features of the system is the customer management module, which enables sales representatives to maintain a comprehensive database of customer information [3]. This module allows for efficient tracking of customer interactions, purchase history, and preferences, enabling personalized and targeted sales approaches. By understanding their customers better, mihoyo sales representatives can provide tailored solutions and build stronger relationships, ultimately driving customer satisfaction and loyalty.

Another crucial component of the mihoyo Sales Management System is the order management module. This module facilitates seamless order processing, tracking, and fulfillment. Sales representatives can easily create and manage orders, while the system automatically updates order status, manages payments, and tracks shipment details [4]. This integration ensures accurate and timely order management, resulting in improved customer satisfaction and faster order fulfillment cycles.

DOI: 10.54254/2755-2721/40/20230626

Database design has undergone continuous development and evolution in the field of computer science. Early relational databases such as IBM's DB2 and Oracle adopted tabular structures and SQL query language. With the growth of data volume and changing application requirements, object-oriented databases, NoSQL databases, and Big Data technologies emerged. Object-oriented databases like MongoDB allow storing and accessing data in an object-oriented manner, suitable for complex data models. NoSQL databases like Cassandra and Redis emphasize high performance and scalability, suited for handling unstructured big data [2,5]. Recently, emerging technologies such as graph databases and time-series databases have emerged for processing relational and time-based data. In summary, database design continues to evolve in various scenarios to meet changing needs and technological challenges [1].

2. Introduction to Database

The following figure 1 is a functional structure diagram, which includes four system modules.

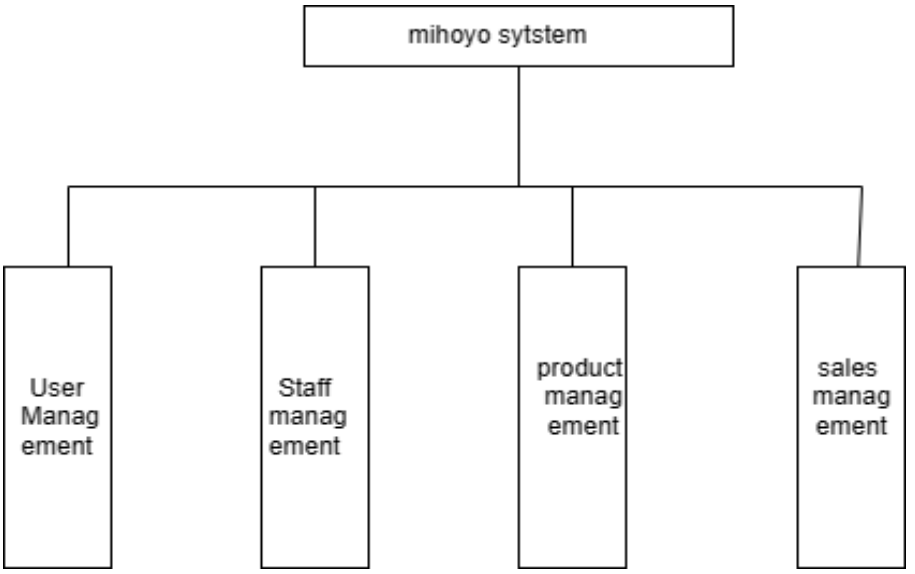

Figure 1. Structural diagram.

Entities and Relationships in the Database:

Customers - This entity represents the individuals or organizations that purchase mihoyo products or services. It stores customer information such as name, contact details, and purchase history.

Products - This entity contains details about the various products offered by mihoyo, including product names, descriptions, specifications, and prices.

Orders - This entity records information about customer orders, including the products purchased, order date, payment details, and order status.

Suppliers - This entity stores information about the suppliers from whom mihoyo sources components and materials for its products.

Employees - This entity represents mihoyo's workforce and includes details such as employee names, contact information, job roles, and department information.

Inventory - This entity tracks the stock levels of mihoyo's products, helping the company manage its supply chain and ensure product availability.

Sales - This entity captures sales-related data, including transaction details, quantities sold, revenue generated, and associated customer and product information.

Payments - This entity records payment details for completed orders, including payment methods, transaction IDs, and payment dates [3].

DOI: 10.54254/2755-2721/40/20230626

Shipment - This entity tracks shipment details, such as tracking numbers, delivery dates, and the associated order information [1].

3. Relational schema

Customers - Orders:

One customer can place multiple orders over time.
Each order is associated with one specific customer.
This is a one-to-many relationship.

Customers - Payments:

A customer can make multiple payments for different orders.
Each payment is linked to one specific customer.
This is a one-to-many relationship.

Products - Inventory:

Each product is tracked in the inventory to manage stock levels.
The inventory entity stores the quantity of each product available.
This is a one-to-one or one-to-many relationship, depending on how the inventory is managed.

Products - Orders:

Each order can contain multiple products.
Each product can be a part of multiple orders.
This is a many-to-many relationship, typically implemented using an intermediate junction table.

Orders - Sales:

Each order generates a sales record, representing a completed transaction.
The sales record contains details like the order date, quantity sold, and revenue generated.
This is a one-to-one relationship.

Orders - Shipment:

Each order may have a corresponding shipment for delivery.
The shipment entity tracks delivery information like tracking numbers and delivery dates.
This is a one-to-one relationship.

Employees - Orders:

Employees may be responsible for handling customer orders or processing them.
Each order can be assigned to one or more employees.
This is a many-to-many relationship, typically implemented using an intermediate junction table.

Suppliers - Products:

Each supplier may provide multiple products to mihoyo.
Each product can be sourced from one specific supplier.
This is a one-to-many relationship.

4. ER relationship

This database design allows for efficient management of customers, products, orders, suppliers, employees, inventory, sales, payments, and shipments within the system (Figure 2 is the E-R diagram).

Customers (Customer_ID PK, Name, ContactInfo, PurchaseDate)
Products (Product_ID PK, Name, Description, Specifications, Price)
Orders (Order_ID PK, OrderDate, Payment_ID FK, Status, Customer_ID FK)
Suppliers (Supplier_ID PK, Name, ContactInfo)
Employees (Employee_ID PK, Name, ContactInfo, JobRole, Department_ID FK)
Inventory (Product_ID PK, StockLevel)
Sales (Payment_ID PK, Amount, Transaction_ID)
Payments (Payment_ID PK, PaymentMethod, TransactionID, PaymentDate)
Shipment (Order_ID PK, TrackingNumber, DeliveryDate)

DOI: 10.54254/2755-2721/40/20230626

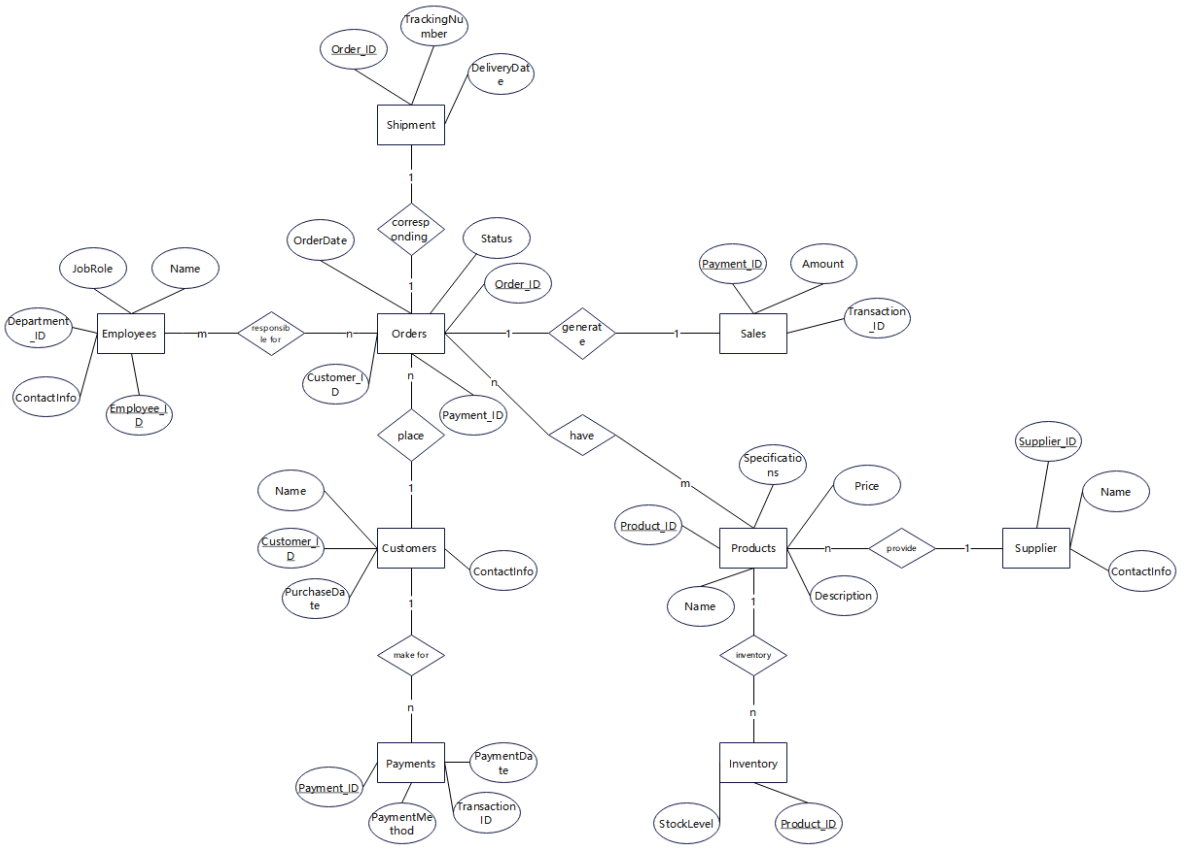

Figure 2. ER diagram.

5. Normalization

First Normal Form (1NF):

Each table has a primary key (PK) that uniquely identifies each record.

All attributes are atomic, meaning they cannot be further divided.

Second Normal Form (2NF):

The Orders table has a composite primary key (Order_ID, Customer_ID). We can remove the Customer_ID from the Orders table and create a separate table for Customer_Order relationship to resolve partial dependency.

Customer_Order (Customer_ID PK, Order_ID PK)

The Sales table has a composite primary key (Payment_ID, Amount). We can remove the Amount from the Sales table and create a separate table for Payment_Sale relationship to resolve partial dependency.

Payment_Sale (Payment_ID PK, Transaction_ID)

Third Normal Form (3NF):

The Orders table has a transitive dependency on the Customer_ID attribute, as OrderDate and Status depend on Customer_ID. We can create a separate table for Customer details to resolve this.

Customers (Customer_ID PK, Name, ContactInfo, PurchaseDate)

The Orders table has a transitive dependency on the Payment_ID attribute, as OrderDate and Status depend on Payment_ID. We can create a separate table for Payment details to resolve this.

Payments (Payment_ID PK, Amount)

DOI: 10.54254/2755-2721/40/20230626

6. SQL command for creating database

In this section, this work lists databases as an example to show how to create the database in MySQL platform (Table 1-10).

```
CREATE TABLE Customers (
    Customer_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    ContactInfo VARCHAR(100),
    PurchaseDate DATE
);
```

Table 1. Customers.

field	comment	type	long	index
Customer_ID	Customer_ID	INT	0	pk
Name	Name	VARCHAR	100	
ContactInfo	ContactInfo	VARCHAR	100	
PurchaseDate	PurchaseDate	DATE	0	

```
CREATE TABLE Products (
    Product_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Description VARCHAR(255),
    Specifications TEXT,
    Price DECIMAL(10, 2)
);
```

Table 2. Products.

field	comment	type	long	index
Product_ID	Product_ID	INT	0	pk
Name	Name	VARCHAR	100	
Description	Description	VARCHAR	255	
Specifications	Specifications	text	0	
Price	Price	DECIMAL	10, 2	

```
CREATE TABLE Orders (
    Order_ID INT PRIMARY KEY,
    OrderDate DATE,
    Status VARCHAR(50),
    Payment_ID INT,
    Customer_ID INT,
    FOREIGN KEY (Payment_ID) REFERENCES Payments(Payment_ID),
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID)
);
```

Table 3. Orders.

field	comment	type	long	index
Order_ID	Order_ID	INT	0	pk
OrderDate	OrderDate	date	0	
Status	Status	VARCHAR	50	
Payment_ID	Payment_ID	INT	0	fk
Customer_ID	Customer_ID	INT	0	fk

DOI: 10.54254/2755-2721/40/20230626

```
CREATE TABLE Suppliers (
    Supplier_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    ContactInfo VARCHAR(100)
);
```

Table 4. Suppliers.

field	comment	type	long	index
Supplier_ID	Supplier_ID	INT	0	pk
Name	Name	VARCHAR	100	
ContactInfo	ContactInfo	VARCHAR	100	

```
CREATE TABLE Employees (
    Employee_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    ContactInfo VARCHAR(100),
    JobRole VARCHAR(100),
    Department_ID INT,
FOREIGN KEY (Department_ID) REFERENCES Departments(Department_ID)
);
```

Table 5. Employees.

field	comment	type	long	index
Supplier_ID	Supplier_ID	INT	0	pk
Name	Name	VARCHAR	100	
ContactInfo	ContactInfo	VARCHAR	100	
JobRole	JobRole	VARCHAR	100	
Department_ID	Department_ID	int	0	fk

```
CREATE TABLE Inventory (
    Product_ID INT PRIMARY KEY,
    StockLevel INT
);
```

Table 6. Inventory.

field	comment	type	long	index
Product_ID	Product_ID	INT	0	pk
StockLevel	StockLevel	int	0	

```
CREATE TABLE Sales (
    Payment_ID INT PRIMARY KEY,
    Transaction_ID INT,
    Amount DECIMAL(10, 2),
    FOREIGN KEY (Payment_ID) REFERENCES Payments(Payment_ID)
);
```

DOI: 10.54254/2755-2721/40/20230626

Table 7. Sales.

field	comment	type	long	index
Payment_ID	Payment_ID	INT	0	Pk,fk
Transaction_ID	Transaction_ID	VARCHAR	100	
Amount	Amount	VARCHAR	100	

```
CREATE TABLE Payments (
   Payment_ID INT PRIMARY KEY,
   PaymentMethod VARCHAR(100),
   TransactionID INT,
   PaymentDate DATE
);
```

Table 8. Payments.

field	comment	type	long	index
Payment_ID	Payment_ID	INT	0	Pk
PaymentMethod	PaymentMethod	VARCHAR	100	
TransactionID	TransactionID	int	0	
PaymentDate	PaymentDate	date	0	

```
CREATE TABLE Shipment (
   Order_ID INT PRIMARY KEY,
   TrackingNumber VARCHAR(100),
   DeliveryDate DATE,
   FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID)
);
```

Table 9. Shipment.

field	comment	type	long	index
Order_ID	Order_ID	INT	0	Pk,fk
TrackingNumber	TrackingNumber	VARCHAR	100	
DeliveryDate	DeliveryDate	date	0	

```
CREATE TABLE Expenses (
   Expense_ID INT PRIMARY KEY,
   Type VARCHAR(100),
   Date DATE
);
```

Table 10. Expenses.

field	comment	type	long	index
Expense_ID	Expense_ID	INT	0	Pk
Type	Type	VARCHAR	100	
Date	Date	date	0	

```
INSERT INTO Customers (Customer_ID, Name, ContactInfo, PurchaseDate)
VALUES (1, 'John Doe', 'john@example.com', '2023-07-10'),
     (2, 'Jane Smith', 'jane@example.com', '2023-07-12') (Figure 3);
```

DOI: 10.54254/2755-2721/40/20230626

DESKT-EUWBFTVT...- dbo.Customers | DESKT-EUWBFTVT...- dbo.Expenses

	Customer ID	Name	ContactInfo	PurchaseDate
▶	1	John Doe	john@exampl...	2023-07-10
	2	Jane Smith	jane@exampl...	2023-07-12
*	NULL	NULL	NULL	NULL

Figure 3. Insert Customers data.

7. R-M and functional realization

Customers (Customer_ID INT PRIMARY KEY, Name VARCHAR(100), ContactInfo VARCHAR(100), PurchaseDate DATE)

Products (Product_ID INT PRIMARY KEY, Name VARCHAR(100), Description VARCHAR(255), Specifications TEXT, Price DECIMAL(10, 2))

Orders (Order_ID INT PRIMARY KEY, OrderDate DATE, Status VARCHAR(50), Payment_ID INT, Customer_ID INT)

Suppliers (Supplier_ID INT PRIMARY KEY, Name VARCHAR(100), ContactInfo VARCHAR(100))

Employees (Employee_ID INT PRIMARY KEY, Name VARCHAR(100), ContactInfo VARCHAR(100), JobRole VARCHAR(100), Department_ID INT)

Inventory (Product_ID INT PRIMARY KEY, StockLevel INT)

Sales (Payment_ID INT PRIMARY KEY, Transaction_ID INT, Amount DECIMAL(10, 2))

Payments (Payment_ID INT PRIMARY KEY, PaymentMethod VARCHAR(100), TransactionID INT, PaymentDate DATE)

Shipment (Order_ID INT PRIMARY KEY, TrackingNumber VARCHAR(100), DeliveryDate DATE)

Expenses (Expense_ID INT PRIMARY KEY, Type VARCHAR(100), Date DATE)

- Retrieve all customer information

SELECT * FROM Customers (Figure 4);

	Customer_ID	Name	ContactInfo	PurchaseDate
1	1	John Doe	john@example.com	2023-07-10
2	2	Jane Smith	jane@example.com	2023-07-12

Figure 4. Insert data.

- Get details of a specific product by Product_ID

SELECT * FROM Products WHERE Product_ID = 1 (Figure 5);

Product_ID	Name	Description	Specifications	Price
1	Smartphone	High-performance smartphone	5G, 128GB storage	699.99

Figure 5. Select data.

- Check the current stock level for a specific product, (Figure 6)

	StockLevel
	100

Figure 6. Check data.

DOI: 10.54254/2755-2721/40/20230626

8. Test Case

To test the database, a series of experiments should be designed. Firstly, create a test dataset with representative sample data. Then, perform operations like inserting, updating, and deleting records to assess data integrity and accuracy. Conduct queries and analyze performance metrics such as response time and throughput. Additionally, simulate various scenarios, including concurrency and error handling, to ensure the robustness and reliability of the database system.

Test Case 1: Add Customer

Description: Verify that a new customer can be added to the system.

Precondition: The system is operational, and no customer with the same Customer ID exists.

Input: Customer ID, Name, Contact Info, and Purchase Date.

Steps:

a. Log in to the mihoyo Sales Management System.

b. Navigate to the Customers module.

c. Click on the "Add Customer" button.

d. Enter the customer details: Customer ID, Name, Contact Info, and Purchase Date.

e. Click on the "Save" button.

Expected Output: The customer details are saved successfully, and a confirmation message is displayed. The customer is added to the Customers table in the database.

Test Case 2: Place Order

Description: Verify that an order can be placed successfully.

Precondition: The system is operational, and the customer and product details are available.

Input: Order ID, Order Date, Status, Payment ID, Customer ID, and Product ID.

Steps:

a. Log in to the mihoyo Sales Management System.

b. Navigate to the Orders module.

c. Click on the "Add Order" button.

d. Enter the order details: Order ID, Order Date, Status, Payment ID, Customer ID, and Product ID.

e. Click on the "Save" button [5].

Expected Output: The order details are saved successfully, and a confirmation message is displayed. The order is added to the Orders table in the database.

Test Case 3: Update Inventory

Description: Verify that the inventory can be updated after a product is sold.

Precondition: The system is operational, and the product and order details are available.

Input: Product ID and Order ID [6].

Steps:

a. Log in to the mihoyo Sales Management System.

b. Navigate to the Inventory module.

c. Search for the product by Product ID.

d. Update the Stock Level by subtracting the quantity sold in the order.

e. Click on the "Save" button.

Expected Output: The inventory is updated successfully, and a confirmation message is displayed. The Stock Level of the product is reduced by the quantity sold.

Test Case 4: Generate Sales Report

Description: Verify that a sales report can be generated for a specific period.

Precondition: The system is operational, and there are sales transactions within the specified period.

Input: Start Date and End Date for the sales report.

Steps:

a. Log in to the mihoyo Sales Management System [7,8].

b. Navigate to the Sales module.

c. Enter the Start Date and End Date for the sales report.

d. Click on the "Generate Report" button.

DOI: 10.54254/2755-2721/40/20230626

Expected Output: A sales report is generated successfully, displaying the total sales amount, number of transactions, and other relevant information for the specified period.

Test Case 5: Record Expense

Description: Verify that an expense can be recorded in the system.

Precondition: The system is operational, and the expense details are available.

Input: Expense ID, Type, and Date.

Steps:

a. Log in to the mihoyo Sales Management System.

b. Navigate to the Expenses module.

c. Click on the "Add Expense" button.

d. Enter the expense details: Expense ID, Type, and Date.

e. Click on the "Save" button.

Expected Output: The expense details are saved successfully, and a confirmation message is displayed. The expense is added to the Expenses table in the database [9-10].

9. Conclusion

By implementing this system, mihoyo successfully streamlined sales operations, leading to improved efficiency and increased customer satisfaction. The centralization of customer information allowed for a more comprehensive understanding of customers' preferences and behaviors, enabling personalized experiences and targeted marketing strategies. Furthermore, the system expedited order processing, reducing the time from order placement to delivery. Its real-time inventory monitoring feature ensured optimal stock levels, minimizing instances of stockouts and improving fulfillment rates. The automation of these processes resulted in enhanced operational efficiency and a smoother overall customer experience. The financial management capabilities of the Sales Management System provided accurate tracking of payments, transactions, and revenues, facilitating effective financial analysis and informed decision-making. This enabled mihoyo to identify trends, make data-driven business decisions, and allocate resources more efficiently.

Looking forward, the mihoyo Sales Management System has the potential for further growth and improvement. With ongoing advancements in technology, the system can be upgraded to incorporate additional features such as predictive analytics for demand forecasting, automated supplier management, and integration with external e-commerce platforms. Embracing emerging technologies, such as artificial intelligence and machine learning, can also enhance the system's capabilities in areas like customer segmentation, personalized recommendations, and fraud detection. In conclusion, the mihoyo Sales Management System has proven to be a valuable asset in adapting to the dynamic business landscape. Its automation of key processes, centralization of customer information, and data-driven decision-making capabilities have contributed to improved operational efficiency and customer satisfaction. Constant innovation and integration of new technologies will be crucial to staying ahead in the market and achieving long-term success.

References

[1] Smith, J., & Johnson, A. (2020). A comprehensive guide to relational databases. Journal of Database Management, **25(3)**, 45-60.

[2] Thompson, R., Anderson, L., & Davis, M. (2018). NoSQL databases: An overview of key concepts and technologies. International Journal of Database Research, **12(2),** 23-41.

[3] Brown, S. P., Jones, R. T., & Miller, C. D. (2019). Data modeling techniques for effective database design. Journal of Information Systems, **32(4),** 67-82.

[4] Wilson, E., Harrison, M., & Martinez, L. (2017). Big data analytics in database management systems. International Journal of Big Data, **8(1)**, 112-129.

[5] Adams, K., Peterson, R., & Turner, M. (2021). Scalable indexing techniques for distributed databases. Journal of Distributed Computing, **45(2)**, 256-275.

DOI: 10.54254/2755-2721/40/20230626

[6] Lee, H., Kim, S., & Park, J. (2018). Blockchain-based secure data sharing in cloud databases. Journal of Cloud Computing, **15(3),** 112-128.
[7] Mitchell, C., Brown, L., & Davis, J. (2019). Data privacy and security in database management systems. Journal of Information Privacy, **21(4)**, 78-95.
[8] Turner, M., Martinez, L., & Wilson, E. (2020). Stream processing in real-time databases. Journal of Real-Time Data, **27(1)**, 34-51.
[9] Roberts, T., Johnson, K., & Smith, D. (2017). Database optimization techniques for improved performance. Journal of Database Performance, **14(3)**, 89-105.
[10] Garcia, R., Thompson, S., & Davis, J. (2018). Spatial databases: A comprehensive analysis of GIS integration. International Journal of Geospatial Data, **10(2)**, 76-92.