INTRODUCTION

SUMMARY

Oberon Investments PLT is an investment and wealth management firm providing multiple financial services. Considering its stock broking service for clients and corporate, a specific use case was created, in which the company plans to develop sub-database dealing with trading service for VIP clients. Sub-database consists of a relational database, which contains data regarding order, customer, bank details, and traders, and a non-relational database, which contains data regarding client product portfolio. There are two use cases, one is customer placing an order and making a bank transaction, and another is customer viewing their product portfolio by simply inputting customer ID.

Considering the use case, SQLite is selected for RDBMS and MongoDB is selected for NoSQL database system. In conceptual database design step, the ER diagram is created to identify entities, the relationship, primary key and the multiplicity constraints. Based on ER diagram, the logical database design is conducted in 3rd Normal form. After modelling database design, relational database is created on SQLite and NoSQL database is created on MongoDB through Jupyter notebook. Considering the context of the user case, polyglot persistence is implemented with CRUD function in Jupyter notebook. This process creates convenient access for the user to select drop down box for their desired order, which is reflected to the database. Moreover, It allows the user to view their product portfolio by simply selecting their customer ID.

OBJECTIVES

- ✓ Develop sub-database to help the company to avoid database performance degradation and enable the user to quickly access trading service.
- ✓ Analyse various RDBMS system, such as SQLite and Microsoft Access, and justify which one better suit the business case. Compare a variety of NoSQL database system, such as Redis, MongoDB, and Neo4j.
- ✓ Implement real-time business function using RDBMS and NoSQL database system to simplify the complex process of data gathering and analysis.
- ✓ Implement polyglot persistence in Jupyter notebook to integrate and analyse various databases to help the organization with critical decision making with data.

ABOUT THE ORGANIZATION

Oberon Investments Group Plc is a financial boutique providing personalized investment and wealth management services for clients, as well as corporate broking service for small or mid-cap companies. The company is established in 2017, with headquarters located in London. Oberon's business strategy is to provide diversified services in wealth management and financial service with multiple revenue streams to offer competitive fee structure. According to the annual report, the company achieved 237% revenue growth to £3.8m for the year ended 31 March 2021.

The business function of this company consists of three divisions: Investment management, Corporate Advisory & Broking, and Wealth Management. The investment management division offers bespoke client portfolios with a variety of services, such as Tax-efficient Investment solutions, fundraising opportunities, Execution-only services, etc. The corporate advisory & broking division accesses to capital and investors and provides strategic advice and corporate services. The wealth management division delivers advice regarding wealth management within a club-style environment, which facilitates networking. The service includes cash flow modelling, financial planning, mortgages, corporate finance, etc.

BUSINESS RULES

Oberon Investments Group Plc runs a main database. Due to the large amount of data they contain, the data volume has dramatically increased and the access performance has become slow. To avoid database performance degradation caused by large main database, Oberon Investments would like to create more simplified sub-database dealing with trading service for VIP clients with recommended trading product. This will allow the user to make quick transactions and quickly access the database. And the main database will operate for generating non-trading service statement, IPO, Corporate action, trading day setup, settlement process, and market charge, etc.

Sub-database consists of one relational database and NoSQL database, which interacts each other. The relational database consists of data regarding customer profile, trader, product, customer's fund id and order details. There are a total of 10 products, which will be updated according to trader's recommendation on daily or weekly basis. The NoSQL database contains client product portfolio data, which will be updated when customer order is to be approved.

The transaction has two business days settlement (T+2) after the order executes, so client's account balance will be updated after two days. In the case that client sells the product within two days, the transaction will be on hold and executed after it is updated. Due to the cost concern and regulation which only allows main database to receive current market value from stock exchange, the system of sub-database is restricted from updating the source of up-to-date market value.

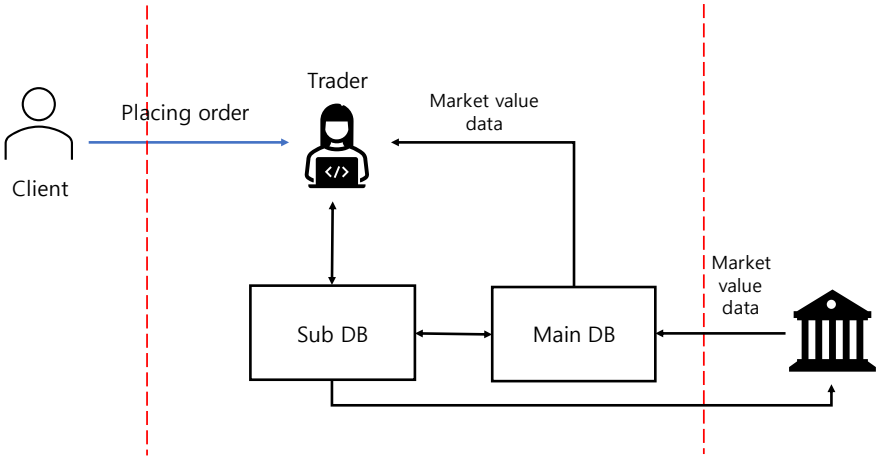


Figure 1 - Database structure

USE CASE

USE CASE 1

The following use case describes client's order placing and transaction process.

Order Placing and Transaction

Main Success Scenario:

1. The user (customer) places an order.
2. The user input customer id, product id, whether buy or sell, price, quantity and fund id into the system.
3. The system reads the user's fund account and authorizes the order if the balance is sufficient.
4. They system updates the user's order status to approved.
5. The user makes bank transactions.
6. The user's bank transaction is approved.

Extensions:

2a. The user doesn't input one or more mandatory fields (customer id, product id, buy or sell, price, quantity, and fund id)

.1 The system displays an error message to ask the user to fill out all mandatory fields.

3a. The system unauthorize the order since the user's fund account balance is insufficient.

.1 The system updates user's order status to rejected.

USE CASE 2

The following use case describes client selling the product and updating their product portfolio.

Enquiring client product portfolio

Main Success Scenario:

1. The user (customer) enquires their product portfolio to view.
2. The user input customer id.
3. The system read trade id, product id and quantity according to customer id.
4. The system displays client's each product and quantity by visualization.

Extensions:

3.a. When the customer wants to view certain product, they input customer id and product id they would like to view.

.1 The system will display only required product and its quantity.

CHOICE OF DATABASE

SQLite for RDBMS

SQLite is embedded, public-domain, server-less relational database engines. It is an in-memory library, with zero-configuration needed. Microsoft Access is another database management system from Microsoft, which combines the relational database engine with a graphical user interface. Considering the use case of the company, it is ideal choose SQLite for Relational Database system. The reason of choosing SQLite is followed:

1. Server operating systems – SQLite is serverless, so that the company can run and store database on any operating platforms. On the other hand, Microsoft Access can only be used on Windows.
2. Supported languages – SQLite works with most of every programming language while Access supports only a few languages, such as C, Java, VBA, etc. As Python will be used to perform CRUD functions of persistent storage, it is more suitable to use SQLite which supports Python.
3. Considering that Oberon Investments run multiple databases, SQLite can facilitate the flexibility of database by working on the same session simultaneously.

MongoDB for NoSQL database system

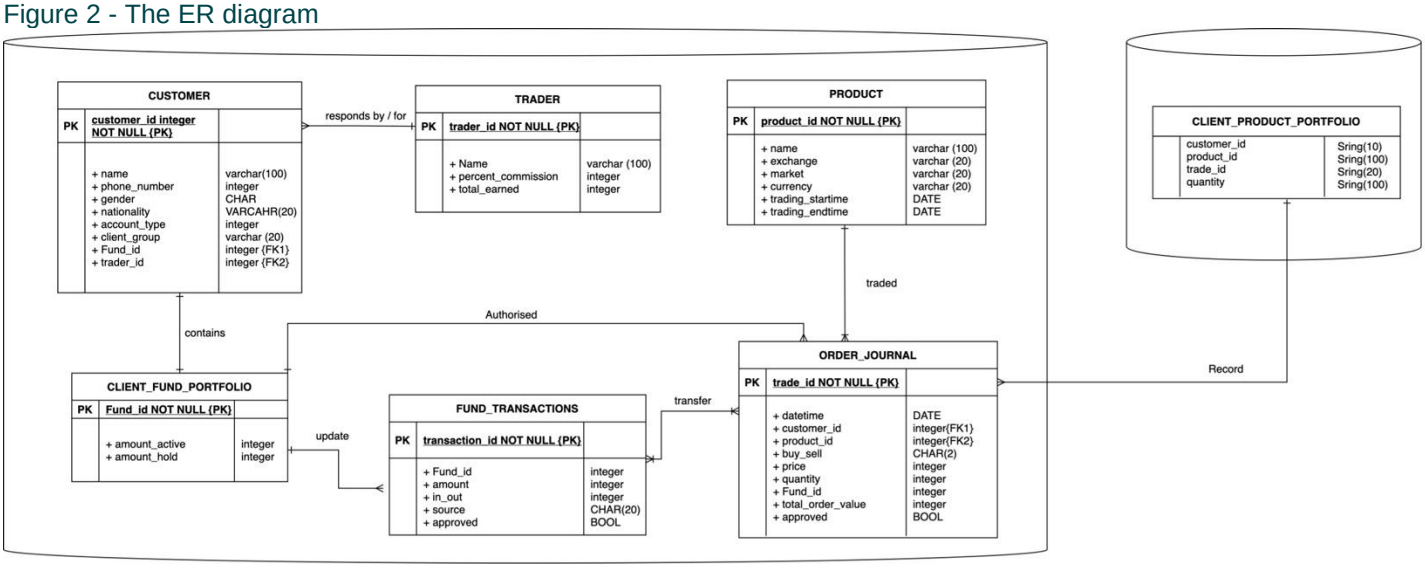
For the considered use case, the ideal option of NoSQL database system is MongoDB. The reason for choosing MongoDB is listed below:

1. It is schema-less – The data that needs to be inserted in the DB need not possess a predefined structure.
2. Size and speed of operation – Using MongoDB, larger amount of data can be stored and processed with at most ease and speed. Further the processing speed of editing the existing database and querying the database frequently can be done frequently with least processing time.
3. The use case under consideration had a wide scope in the future and needs a system that is compatible to extend its memory in the future.
4. It is easy to save unstructured data in MongoDB. Also, the database can be secured either by using simple username and password or complex security tools like RBAC.
5. Best suited for real time analytics applications such as stock market or banking applications and other Internet of Thing's applications using real time data.

DATABASE MODELLING

Conceptual database design

Figure 2 - The ER diagram



Data Dictionary

Table/Collection name	Attribute name	Contents	Type	Format	Domain	Required	PK or FK	FK references
CUSTOMER	customer_id	Id of the customer	varchar (100)	C9		Y	PK	
	Name	Name of the customer	varchar (100)	Xxxx Xxxx		Y		
	phone_number	Phone number of the	integer	0999999999		Y		
	dob	customer	char	F/M		N		
	gender	Gender of the customer	varchar (20)	Xxxxx		N		
	nationality	Nationality of the customer	integer	Individual/ Entity		Y		
	account_type	Client account type	varchar (20)	Xxxx		N		
	client_group	Level of client account	varchar (20)	Xxxx		N		
	Fund_id	Client group	integer	99		Y	FK	CLIENT_FUND_PORTFOLIO (Fund_id)
	trader_id	Id of client fund account	integer	99		Y	FK	TRADER (trader_id)
		Id of trader responsible						
TRADER	trader_id	Id of agents responsible	integer	99		Y	PK	
	Name	Name of trader	varchar (100)	Xxxx		Y		
	percent_commission	% of earing from trade	integer	9.999%		Y		
	total_earned	Amount of total earing	integer	999999999		Y		
FUND_TRANSACTIONS	transaction_id	Id of the transaction	integer	999999		Y	PK	
	Fund_id	Id of the client fund account	integer	99		Y	FK	CLIENT_FUND_PORTFOLIO(Fund_id)
	amount	Amount of the transaction	integer	9999999999		Y		
	in_out	Type of transaction	integer	IN/Out		Y		
	source	Requests generate from	CHAR(20)	User/Market		Y		
	approved	Result of the transaction	BOOL	0(Rejected),1(Approved)		Y		
CLIENT_FUND_ PORTFOLIO	Fund_id	Id of the client fund account	integer	99		Y	PK	
	amount_active	Available balance of the	integer	9999999999		Y		
	amount_hold	account	integer	9999999999		Y		
		Amount of funds on hold						
PRODUCT	product_id	Id of the product	varchar (8)	Xxxx		Y	PK	
	name	Name of the product	varchar (100)	Xxxx		Y		
	exchange	The exchange belong to	varchar (20)	Xxxx		Y		
	market	Product Market	varchar (20)	Xx		Y		
	currency	Currency of the Market	varchar (20)	Xxx		Y		
	trading_startime	Market trading start time	DATE	(Time in string format)		Y		
	trading_endtime	Market Trading end time	DATE	(Time in string format)		Y		
ORDER_JOURNAL	trade_id	Id of the order	integer	9999		Y	PK	
	datetime	Date and Time order create	DATE	(Timestamp in string)		Y		
	customer_id	Id of the customer	varchar (100)	C9		Y	FK	CUSTOMER(customer_id)
	product_id	Id of the product in the order	integer	Xxxx		Y	FK	PRODUCT(product_id)
	buy_sell	Type of order	CHAR(2)	B/S		Y		
	price	Order price input	integer	999,999,999.99		Y		

	quantity	Amount product in the order	integer	999		Y		
	Fund_id	Id of the client fund account	integer	99		Y	FK	CLIENT_FUND_PORTFOLIO(Fund_id)
	total_order_value	Total value involved in the order	integer	999,999,999.99		Y		
	approved	Result of the order request	BOOL	0(Rejected),1(Approved)		Y		
	CCY	Currency of the product	integer	Xxx		Y	FK	PRODUCT (currency)
CLIENT_PRODUCT_ PORTFOLIO	customer_id	Id of the customer	Sring (10)	99999999999		Y		
	product_id	Id of the product	Sring (100)	Xxxx		Y		
	trade_id	Id of the order	Sring (20)	99		Y		
	quantity	Amount of the product	Sring (100)	999		Y		

Logical database design

CUSTOMER (customer_id, Name, phone_number, dob, gender, nationality, account_type, client_group, Fund_id, trader_id)
PRIMARY KEY customer_id
FOREIGN KEY Fund_id REFERENCES CLIENT_FUND_PORTFOLIO (Fund_id)
FOREIGN KEY trader_id REFERENCES TRADER (trader_id)

TRADER (trader_id, Name, percent_commission, total_earned)
PRIMARY KEY trader_id

FUND_TRANSACTIONS (transaction_id, Fund_id, amount, in_out, source, approved)
PRIMARY KEY transaction_id
FOREIGN KEY Fund_id REFERENCES CLIENT_FUND_PORTFOLIO(Fund_id)

CLIENT_FUND_PORTFOLIO (Fund_id, amount_active, amount_hold)
PRIMARY KEY Fund_id

PRODUCT (product_id, name, exchange, market, currency, trading_startime, trading_endtime)
PRIMARY KEY product_id

ORDER_JOURNAL (trade_id, datetime, customer_id, product_id, buy_sell, price, quantity, Fund_id, total_order_value, approved, CCY)
PRIMARY KEY trade_id
FOREIGN KEY customer_id REFERENCES CUSTOMER(customer_id)
FOREIGN KEY product_id REFERENCES PRODUCT(product_id)
FOREIGN KEY Fund_id REFERENCES CLIENT_FUND_PORTFOLIO(Fund_id)
FOREIGN KEY CCY REFERENCES PRODUCT(currency)

Physical database design

Relational part

Below is the SQL code to great the table:

```
CREATE TABLE Client_Fund_Portfolio (
Fund_id INTEGER NOT NULL
PRIMARY KEY
UNIQUE,
amount_active INTEGER,
amount_hold INTEGER
);

CREATE TABLE Customer (
customer_id VARCHAR NOT NULL
PRIMARY KEY
UNIQUE ON CONFLICT ROLLBACK,
name VARCHAR (100) NOT NULL ON CONFLICT FAIL,
phone_number INTEGER (100) NOT NULL ON CONFLICT FAIL,
dob DATE,
Gender CHAR,
nationality VARCHAR (20),
Account_type INTEGER NOT NULL ON CONFLICT ROLLBACK,
Client_group VARCHAR (20),
Fund_id INTEGER REFERENCES Cline_Fund_Portfolio (Fund_id)
NOT NULL,
Trader_id INTEGER REFERENCES Trader (trader_id)
NOT NULL
);


CREATE TABLE Fund_Transactions (
Transaction_id INTEGER PRIMARY KEY
NOT NULL
UNIQUE,
Fund_id INTEGER REFERENCES Cline_Fund_Portfolio (Fund_id),
Amout INTEGER,
in_out INTEGER,
Source CHAR (20),
Approved BOOLEAN
);

CREATE TABLE Order_Journel (
trade_id INTEGER PRIMARY KEY AUTOINCREMENT
NOT NULL
UNIQUE,
datetime TEXT DEFAULT (CURRENT_TIMESTAMP),
customer_id INTEGER REFERENCES Customer (customer_id),
product_id INTEGER REFERENCES Product (Product_id),
buy_sell CHAR (2),
```

```
Price INTEGER,
Quantity INTEGER,
Fund_id INTEGER REFERENCES Cline_Fund_Portfolio (Fund_id),
Total_order_value INTEGER,
Approved BOOLEAN
);

CREATE TABLE Product (
Product_id VARCHAR PRIMARY KEY
NOT NULL
UNIQUE,
Name VARCHAR (100),
exchange VARCHAR (20),
Market VARCHAR (20),
Currency VARCHAR (20),
Trading_starttime DATE,
trading_endtime DATE
);

CREATE TABLE Trader (
trader_id INTEGER PRIMARY KEY
NOT NULL
UNIQUE,
Name VARCHAR (100),
Percent_commission INTEGER,
Total_earned INTEGER
);
```

Non-relational part (document database)

Below is the sample data input:

```
Sample_Product_Record = { " customer_id ": "C9",
                        "product_id ": "Xxxx",
                        " trade_id "99
                        " quantity ": 999"}
```

The non-relational table is useful for obtaining select client-owned products. If the client place order in the system. Then the table in SQL [Fund_Transactions] need to be updated , but also the Product_Owne Mongo DB should be updated.

Not only for purchase credit checking but also for traders to analyse client perchance. By observed the correlation of the data, traders can understand the client's preference for investment and provide valuable suggestions.

Use Case Coding

In [8]:

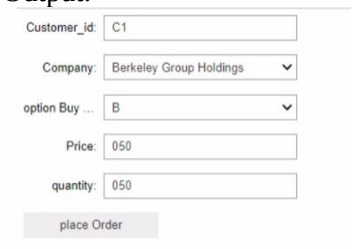
```
#Use case 1

#get the list of companies
query_product = "select product_id, name from product order by name asc"
cur.execute(query_product)
res = cur.fetchall()
list_product = []
list_product.append(("", ""))
for p_id, p_name in res:
    list_product.append((p_name, p_id))


#create the widgets
customer_id_display = widgets.Text(description = 'Customer_id: ')
dropdown_product = widgets.Dropdown(options = list_product, description='Company:')
buy_sell_display = widgets.Dropdown(options = ['','B','S'], description='option Buy (B) / Sell (S):')
price_display = widgets.IntText(description = 'Price: ')
quantity_display = widgets.IntText(description = 'quantity: ')
button = widgets.Button(description="place Order")
output = widgets.Output()

#function to find customer id
def findcustomerid(customer_id):
    sts = f'SELECT customer_id,name FROM customer where customer_id = \'{customer_id}\';'
    cur.execute(sts)
    res = cur.fetchall()
    y = pd.DataFrame(res, columns = ['customer_id', 'name'])
    if y.empty:
        ddd = 0
        return ddd
    else:
        str = y.iloc[0]['name']
        str = str.replace(u'\xa0', u' ')
        return str
```

```
# function to check funds
def checkfunds(funds,customer_id):

    sts = f'SELECT customer_id,fund_id FROM customer where customer_id = \'{customer_id}\';'
    cur.execute(sts)
    res = cur.fetchall()
    y = pd.DataFrame(res, columns = ['customer_id', 'fund_id'])
    fund_id = y.iloc[0]['fund_id']


    # Read function for Relational DB
    sts = f'SELECT amount_active FROM client_fund_portfolio where fund_id = \'{fund_id}\';'
    cur.execute(sts)
    res = cur.fetchall()
    y = pd.DataFrame(res, columns = ['amount_active'])
    yy = y.iloc[0]['amount_active']
    return 1

# function called when button is triggered
def on_button_clicked(b):
    customer_id = customer_id_display.value
    product_id = product = dropdown_product.value
    buy_sell = buy_sell_display.value
    price_value = price_display.value
    quantity_value = quantity_display.value
    funds = price_value * quantity_value
```

```
# function called when button is triggered
def on_button_clicked(b):
    customer_id = customer_id_display.value
    product_id = product = dropdown_product.value
    buy_sell = buy_sell_display.value
    price_value = price_display.value
    quantity_value = quantity_display.value
    funds = price_value * quantity_value

    x = findcustomerid(customer_id)
    if x == 0:
        print("Sorry we could not find your id in our database, please check your id")
    else:
        y = checkfunds(funds,customer_id)

        if y  == 0:
            print("insufficient funds")
        else:
            insertApptStmt = "INSERT INTO order_journel (datetime, customer_id, product_id, " + \
                             "buy_sell, price, quantity, fund_id,total_order_value,approved)" + \
                             "VALUES(?,?,?,?,?,?,?,?,?) "
            cur.execute(insertApptStmt, (datetime.datetime.now(),customer_id, product_id,
                       buy_sell, price_value, quantity_value, y, funds,"Y"))
            print("Your transaction has been successful, updating into your portfolio")
            restul_query = product_own.client_product_portfolio.find({'customer_id': customer_id})
            for r in restul_query:
                print(r)
            results_med_record = client_product_portfolio_access.find({'customer_id': customer_id})
            df_med_records1 = pd.DataFrame(results_med_record, columns = ['customer_id','product_id', 'quantity'])
            display(df_med_records1)
            a = 0
```

```
            for i in df_med_records1['product_id'].values:

                if product_id == i:

                    #qty == df_med_records1.iat[a,2]
                    qty = df_med_records1.get_value(a,"quantity")

                    qty1 = qty + quantity_value
                    qty1 = qty1.item()

                    query = { "product_id": product_id }
                    new_values = { "$set": { "quantity": qty1 } }
                    result_update = client_product_portfolio_access.update_one(query, new_values)
                    break
                a = a+1

            results_med_record = client_product_portfolio_access.find({'customer_id': customer_id})
            df_med_records1 = pd.DataFrame(results_med_record, columns = ['customer_id','product_id', 'quantity'])
            display(df_med_records1)
            plt.bar(df_med_records1.iloc[:,1],df_med_records1.iloc[:,2])
            plt.show()



#Display the buttons
button.on_click(on_button_clicked)
display(customer_id_display)
display(dropdown_product)
display(buy_sell_display)
display(price_display)
display(quantity_display)
display(button, output)
```

Output:

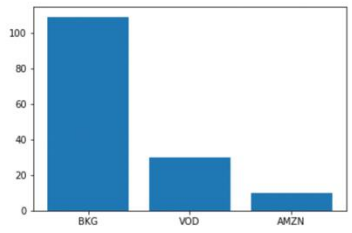



```
Your transaction has been successful, updating into your portfolio
{'_id': ObjectId('62da5194f900cefeee3010b2'), 'customer_id': 'C1', 'product_id': 'BKG', 'quantity': 59}
{'_id': ObjectId('62da5194f900cefeee3010b3'), 'customer_id': 'C1', 'product_id': 'VOD', 'quantity': 30}
{'_id': ObjectId('62da5194f900cefeee3010b4'), 'customer_id': 'C1', 'product_id': 'AMZN', 'quantity': 10}
```

	customer_id	product_id	quantity
0	C1	BKG	59
1	C1	VOD	30
2	C1	AMZN	10

```
E:\Anaconda\lib\site-packages\ipykernel_launcher.py:92: FutureWarning: get_value is deprecated and will be removed in a future release. Please use .at[] or .iat[] accessors instead
```

	customer_id	product_id	quantity
0	C1	BKG	109
1	C1	VOD	30
2	C1	AMZN	10



CONCLUSION

Through this database project, real-time business functions and use cases of Oberon Investment PLT were identified, and simplified sub-database was configured from complex database. Based on the existing stockbroker organization, multiple database technologies were used to create feasible solutions for the organization's use case. We constructed database structure and data requirements by taking into account business function and operation of the organization, as well as trading days and regulations. In order to apply the most appropriate database system for the business case, the evaluation for each relational and non-relational database system was conducted, and eventually SQLite and MongoDB were selected.

The modelling followed structured modelling process for relational and NoSQL database parts. First, ER diagram was created to identify the entities, the relationships, and the primary. Based on ER diagram, a Logical Database Design in 3rd Normal form was further developed. After database design process, polyglot persistence was implemented to connect RDMS and NoSQL, in which CRUD function was performed. This enabled to successfully perform analytics operation considering the organization's use case. The user will be able to access their product portfolio by inputting customer ID, place an order and make a transaction. Two databases perform interactively. This will also help the organization to integrate and analyse a variety of multiple databases and enhance data performance with large amount of data.

Reference

MongoDB vs Redis: Complete Comparision in 2022 - Naiveskill. (2021, August 23). https://naiveskill.com/mongodb-vs-redis/

SQLite vs Microsoft Access - A quick comparison. (2018, November 7). TablePlus. https://tableplus.com/blog/2018/11/sqlite-vs-access.html#:~:text=SQLite%20is%20server%2Dless.

Key-Value Database: How It Works, Key Features, Advantages. (n.d.). InfluxData. Retrieved July 20, 2022, from https://www.influxdata.com/key-value-database/

Final Results. (2021, July 29). Oberon Investment Group. https://oberoninvestments.com/investor-area/final-results-2021/

Boyden, D. (2021, February 17). *Our Services*. Oberon Investment Group. https://oberoninvestments.com/our-services/