Northeastern University

College of Engineering

Database Management System with Analytics for a Financial Services Company

A project report submitted

to

NORTHEASTERN UNIVERSITY

COLLEGE OF ENGINEERING

For partial fulfilment of the requirement

Of

IE 6700: Database Management for Analytics

By

Narayana Sudheer Vishal Basutkar *(002728444)*

[redacted]

Under the guidance of

Prof. Venkat Krishnamurthy

Department: Mechanical + Industrial Engineering

Contents

Project idea

NEU, a private bank has decided to move from traditional file based approach to database approach and has asked us to create its database system to efficiently store data and analyse its revenue across the divisions and services that they offer.

NEU is a global firm that offers Wealth Management, Asset Management, Investment Banking, and Personal & Corporate Banking services. The bank has separate divisions each having a Division ID, Division Name, Division Head ID, and Net Revenue.

The bank records the Employee ID, Employee Name, SSN, Address, Contact numbers, Emails, Date of joining, Job Role, and Yearly Compensation of each of its employees.

The Wealth Management (WM) division caters to only High Net-Worth Individuals (HNI) with liquid assets worth more than 1 million dollars. The division appoints a Wealth Manager for each of its clients who work personally for them. One wealth manager can be working for a maximum of 5 clients. He/She can have many employees working under him/her. The wealth manager handles the entire balance sheet of the client and advises on all of their purchases/ gifts, investments, savings, and taxations. They provide tailored advice and investment services like investment management, estate planning, and corporate and finance advice. Investment management service includes portfolio planning, stock trading on behalf of the client, and other investments like mutual funds, securities, and bonds. The wealth manager suggests the services to the client and it is up to the client to decide on what all services he/she would like to take. Wealth managers make money by charging their clients a fixed fee and also taking a 1% cut from their assets.

The Asset Management (AM) division works for common people, and corporates/institutions. Based on the investor's willingness to take risks, asset managers create a portfolio for the clients by investing in stocks, mutual funds, etc. The asset managers charge their clients a fee for the services they offer and take a 5% cut from the individuals' profits.

The Investment Banking (IB) division purchases assets from corporates that hold monetary value, also known as securities, by doing complete due diligence, and then create a market to sell to third parties /investors. Underwriting is defined as conducting research and assessing the degree of risk each entity brings to the table, setting a fair borrowing rate for the investors that includes their fee, and creating a market to sell the securities to the investors. Companies that attempt IPO, contact investment bankers who underwrite the securities. Securities have a limited quantity and a price per

stock. The investors can sell the securities only after a lock-in period that is decided during the underwriting process. The date of purchase and quantity of stocks is recorded for each investor. Investment bankers make money by charging fees for underwriting.

The Personal and Corporate (P&C) banking division is for ordinary individuals' everyday banking needs. They provide services like opening and closing accounts, accepting deposits, giving withdrawals, debit cards, and online and mobile banking services. They also give out loans and credit cards and set limits after assessing the credit scores of the individuals. This division earns its profits by setting high loan interest rates and low savings interest rates. For the sake of this project, only the net revenue is recorded and not individual loans and savings of commoners.

The bank wants to save all of this information in its database system to analyse the revenue of each division to make informed decisions. We as data engineers/analysts are tasked to create the database system and provide analytical feedback.

EER Rules

1) Employees are specialised as Wealth Managers, Asset Managers, Investment Bankers, and Personal Bankers with partial and disjoint rule as there are other types of employees and one type of employee cannot be the other.

2) There are only 3 kinds of clients - HNI, Commoners and Corporates.

3) Commoners and Corporates are categorised as Asset Management Client with partial rule as not all not be AM clients.

4) All divisions provide services and services can be specialised as Wealth Management services, Asset Management services and Personal Banking services. There can be other types of services and there can be an overlap of services between the divisions.

5) The Wealth Managers provide WM services for a fee and also charge the clients a wealthcut.

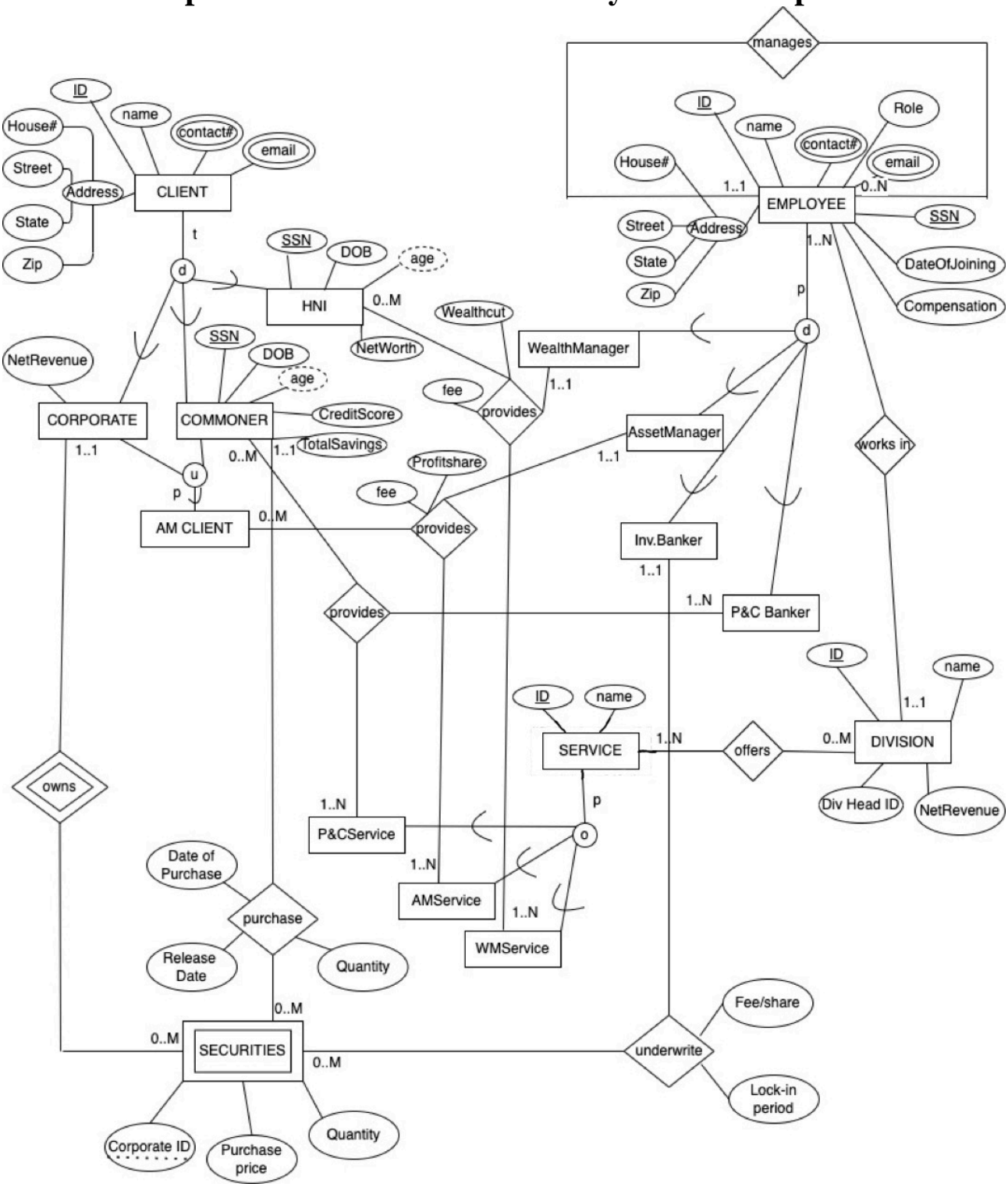
6) Asset Managers provide AM services for a fee and take a profit share for each of the services.

7) Corporates own assets that have monetary value known as securities. Securities are weak entity type as they are existential dependent of the corporate company and they do not have primary key of their own.

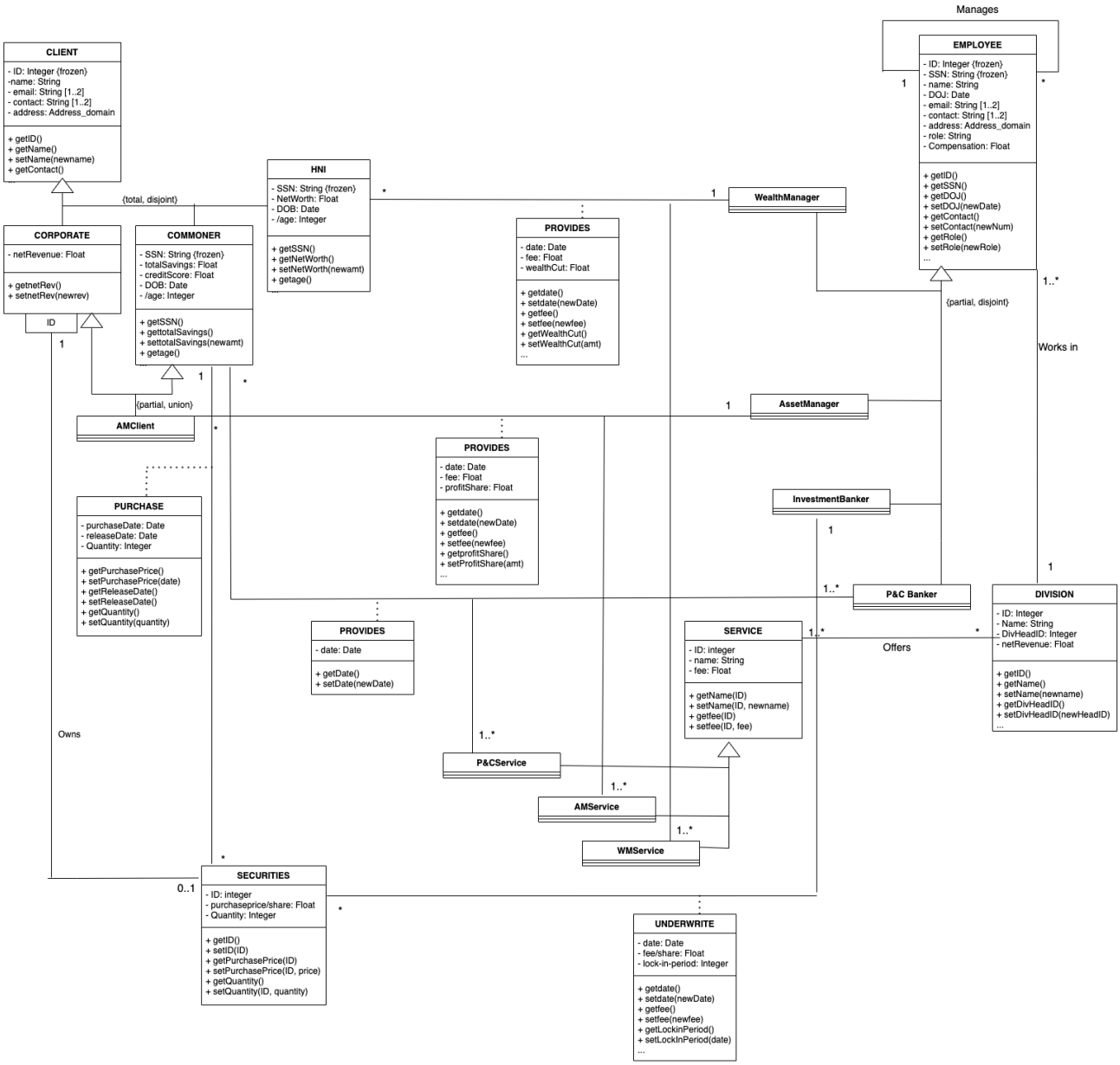
8) Investment Bankers underwrite the securities for a fee per share and also decide on the lock in period during underwriting.

9) Commoners purchase the securities.

Conceptual Model - Enhanced Entity Relationship Model



Conceptual Model - Unified Modelling Language



Link to the image for better picture quality- ***here***

Relational Model

The following are the relations we get after mapping from the EER model. They are not normalised.

Note: Primary keys are underlined, Foreign Keys are in italics, Multi-valued attribute types are in curly braces, and composite attribute types are in parenthesis.

Division (ID, name, divheadID, month, revenue)
Service (ID, name, fees)
Div-Service-Offering (*divID*, *serviceID*) #divID and ServiceID are FK and NOT NULL

WMservice (*serviceID*) AMservice (*serviceID*) PCservice (*serviceID*)

#ServiceID is FK and NOT NULL #ServiceID is FK and NOT NULL #ServiceID is FK and NOT NULL

Employee (ID, SSN, name, role, date-of-joining, {contact#}, {email}, compensation, Address(house#, street, state, zip), *managerID*, *divID*) #Address is composite attribute type and divID is NOT NULL

Client (ID, name, Address(house#, street, state, zip), {contact#}, {email}) #Address is composite attribute type

HNI (*clientID*, SSN, date-of-birth, net worth, age) #clientID is FK and NOT NULL
Commoner (*clientID*, SSN, date-of-birth, age, total savings, creditscore) #clientID is FK and NOT

NULL

CorporateCo (*clientID*, netrevenue) #clientID is FK and NOT NULL

AMclient (*clientID*) #clientID is FK and NOT NULL

WealthManager (*employeeID*) #employeeID is FK and NOT NULL

WMprovision (*HNIclientID*, *WMserviceID*, fee, date, *WMemployeeID*, wealthcut) #HNIclientID, WMemployeeID and WMserviceID are NOT NULL

AssetManager (*employeeID*) #employeeID is FK and NOT NULL

AMprovision(*AMclientID*, *AMserviceID*, fee, profitshare, date, *AMemployeeID*)

#AMclientID, AMserviceID, AMemployeeID are FKs and NOT NULL

PCbanker (*employeeID*) #employeeID is FK and NOT NULL

PCprovision (*commonerClientID*, *PCserviceID*, *PCemployeeID*, date) #commonerClientID, PCserviceID, PCemployeeID are FK and NOT NULL

InvBanker (*employeeID*) #employeeID is FK and NOT NULL

Securities (*CorporateCoClientID*, SecurityID, price-per-share, total-quantity)
#CorporateCoClientID, SecurityID are FKs and NOT NULL

Security-purchase (*commonerClientID*, *SecurityID*, date-of-purchase, quantity, release-date)
#commonerClientID, SecurityID are FKs and NOT NULL

Underwriting (*InvBankerEmployeeID*, *SecurityID*, fee-per-share, lock-in-period)
#InvBankerEmployeeID, SecurityID are FKs and NOT NULL

Relations in 4NF form after Normalization-

(Composite attribute types are made atomic, new relations are made for multi-valued attribute types, and where transitive dependencies and multi-fact relations are found.)

Division (ID, name, divheadID) #divheadID is FK referencing ID
Revenue (*divID,* month, revenue) #divID is FK and NOT NULL
Service (ID, name, fees)
Div-Service-Offering (*divID*, *serviceID*) #divID and ServiceID are FK and NOT NULL

WMservice (*serviceID*)

AMservice (*serviceID*)

PCservice (*serviceID*)

#ServiceID is FK and NOT NULL #ServiceID is FK and NOT NULL #ServiceID is FK and NOT NULL

Employee (ID, SSN, name, role, date-of-joining, compensation, house#, street, state, zip,

managerID, *divID*) #divID is NOT NULL

Employee-contact (*employeeID*, contact#) #EmployeeID is FK and NOT NULL

Employee-email (*employeeID*, email) #EmployeeID is FK and NOT NULL

Client (ID, name, house#, street, state, zip)

Client-contact (*clientID*, contact#) #clientID is FK and NOT NULL

Client-email (clientID, email) #clientID is FK and NOT NULL

HNI (*clientID*, SSN, date-of-birth, net worth, age) #clientID is FK and NOT NULL

Commoner (*clientID*, SSN, date-of-birth, age, total savings, creditscore) #clientID is FK and NOT NULL

CorporateCo (*clientID*, netrevenue) #clientID is FK and NOT NULL AMclient (*clientID*) #clientID is FK and NOT NULL WealthManager (*employeeID*) #employeeID is FK and NOT NULL

WMprovision (*HNIclientID*, *WMserviceID*, fee, date) #HNIclientID and WMserviceID are FK and NOT NULL

Wealthcut (*HNIclientID*, *WMemployeeID*, wealthcut) #HNIclientID and WMemployeeID are FK and NOT NULL

WMclient-mgr (*HNIclientID*, *WMemployeeID*) #HNIclientID, WMemployeeID are FK and NOT NULL

AssetManager (*employeeID*) #employeeID is FK and NOT NULL AMprovision(*AMclientID*, *AMserviceID*, fee, profitshare, date) #AMclientID, AMserviceID are FKs and NOT NULL

AMclient-mgr (*commonerClientID*, *AMemployeeID*) #commonerClientID, AMemployeeID are FK and NOT NULL

PCbanker (*employeeID*) #employeeID is FK and NOT NULL
PCprovision (*commonerClientID*, *PCserviceID*, *PCemployeeID*, date) #commonerClientID,

PCserviceID, PCemployeeID are FK and NOT NULL
InvBanker (*employeeID*) #employeeID is FK and NOT NULL

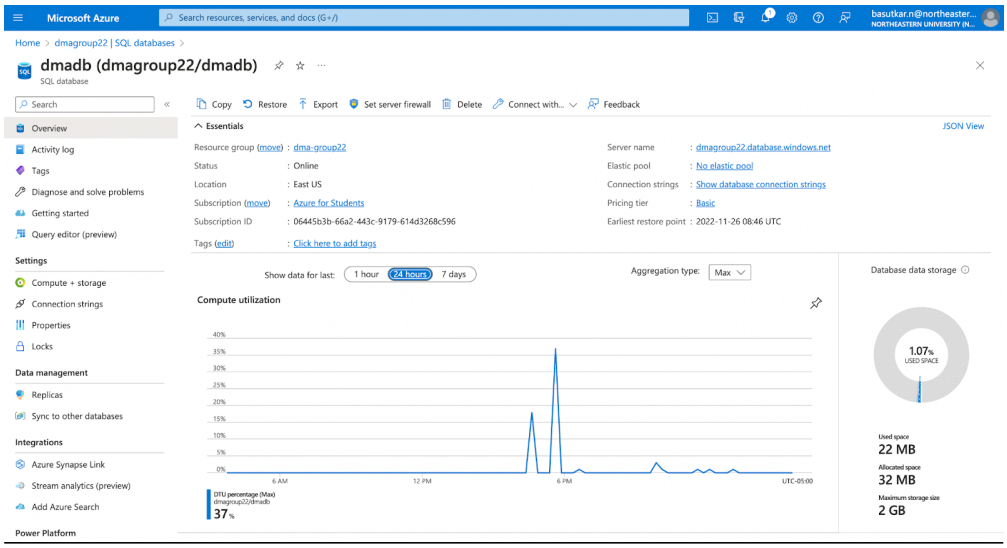
Securities (*CorporateCoClientID*, SecurityID, price-per-share, total-quantity)
#CorporateCoClientID, SecurityID are FKs and NOT NULL

Security-purchase (*commonerClientID*, *SecurityID*, date-of-purchase, quantity, release-date)
#commonerClientID, SecurityID are FKs and NOT NULL

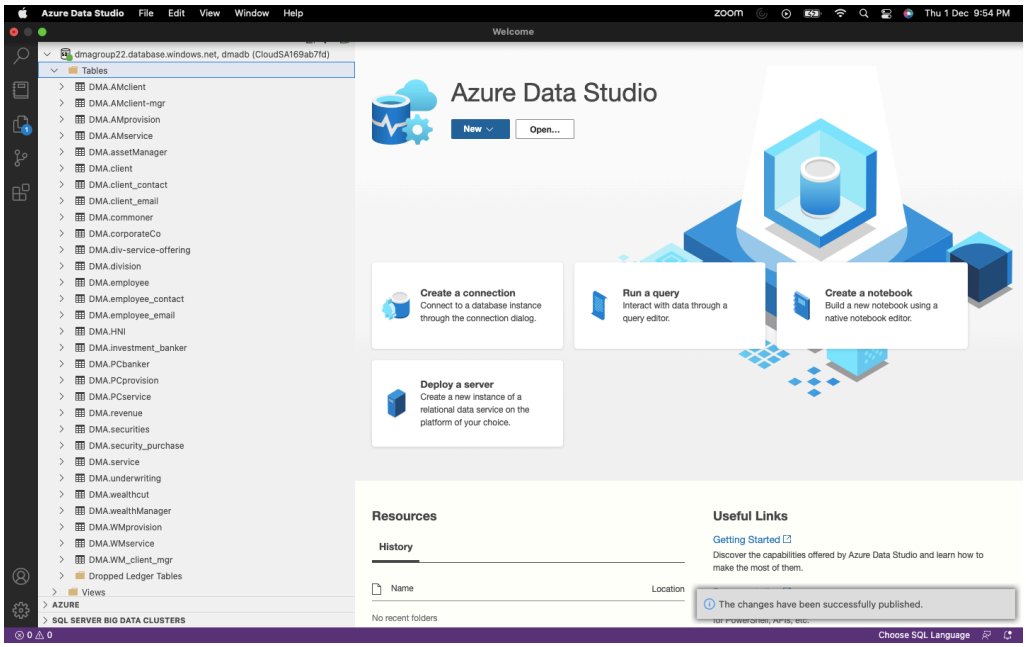
Underwriting (*InvBankerEmployeeID*, *SecurityID*, fee-per-share, lock-in-period)
#InvBankerEmployeeID, SecurityID are FKs and NOT NULL

SQL Queries and their analytical purpose

Since most of the fintech companies are migrating from on-premise Oracle, MySql databases to PostgreSQL, MSSQL databases on Cloud, we implemented our solution in Azure MSSQL with the basic tier of database with 2GB storage. This cloud environment enabled us to work together on the same database using the database server credentials created in the azure portal. IP addresses of our personal computers had to be provided in the firewall rules. **Azure Data Studio** is the cross-platform database tool where we created the schema, tables with foreign key rules, and executed our queries. Below is the screenshot of the schema/tables in the platform.



Azure portal where the DB resource is deployed. We can modify the storage of the database and change/ provide access to users and retrieve the connection string to connect the database to applications from the portal.



Azure Data Studio is the GUI tool to connect to the database, create tables and run queries.

SQL Queries-

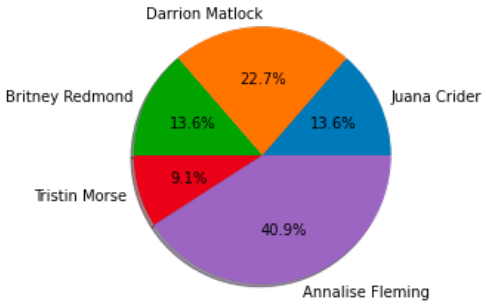
The firm wants to know how each of its division is doing and how much revenue it has accumulated. Revenue of each division is captured differently. First four queries help us gather those insights.

1) Wealth Management: Wealth Managers get a one time wealth cut from the HNI clients and charge fees for each of the services they provide. The revenue will be the sum of the wealth cut and fees. Calculate how much each HNI has paid to the firm.

```
with total_fees as (
select HNI_id, sum(fee) as total_fees_collected
from dma.WMprovision
GROUP by  HNI_id)
select wc.HNI_id, (tf.total_fees_collected + wc.wealthcut) as
total_hni_pay
from dma.wealthcut wc join total_fees tf on wc.HNI_id=tf.HNI_id;
```

Results | Messages

	HNI_id	total_hni_pay
1	1	30014918
2	2	50025008
3	3	30027761
4	4	20015599
5	5	90023574



Created in Python using seaborn library.

2) Asset Management: Asset Managers along with charging fees for their services, also get a profit share from the services they provide. The revenue will be the sum of the fees and profit share. Calculate how much each AM client has paid to the firm. Also show the names of the clients.

```
with total_pay as (
select AMclient_id, sum(profitshare) + sum(fee) as total_amclient_pay
from dma.AMprovision group by AMclient_id)
select tp.AMclient_id, c.name, tp.total_amclient_pay from total_pay tp
join dma.client c on tp.AMclient_id = c.ID;
```

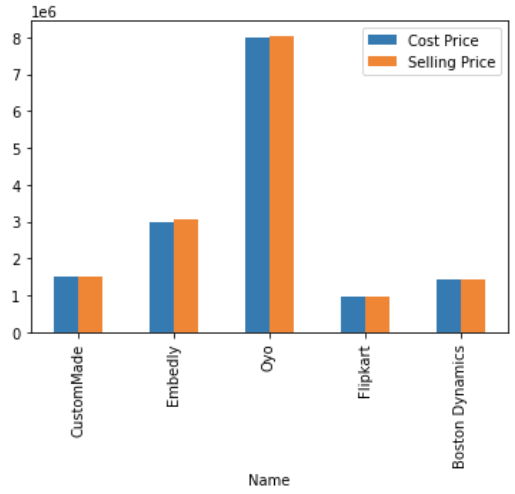
Results | Messages

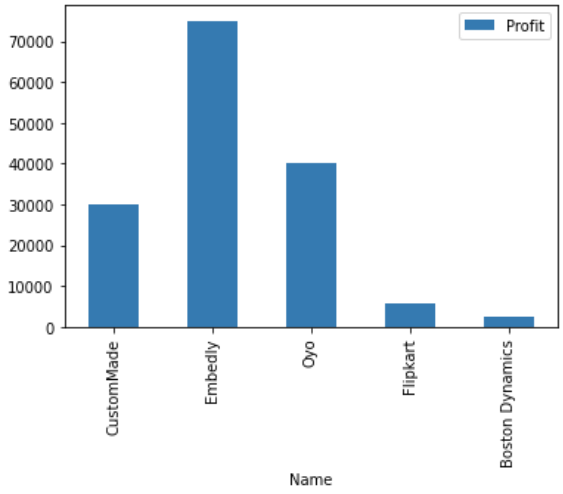
	AMclient_id	name	total_amclient_pay
1	6	Connor Champagne	5877
2	7	Janie Gilman	7367
3	8	Rashad Cerda	5055
4	27	Embedly	6330
5	28	Oyo	8496

3) Investment Banking: Investment Bankers underwrite securities of corporate companies and sell their stocks. They charge a fee/share for underwriting and selling. The revenue brought in by this division will be the (selling price - cost price)* quantity. Assuming all the securities are sold, show the company name, its net revenue, total selling price, total cost price and the profit Investment bankers make on the sale. The total selling price would be (the cost price per share + the underwriting fee per share) * quantity.

```
with IB_stock_sell_price as ( select s.security_id, s.corporateCo_id,
(s.price_per_share + cast(u.fee_per_share as int))*s.total_quantity as
sell_price
from dma.securities s join dma.underwriting u on s.security_id =
u.security_id)
select sp.corporateCo_id, c.name, co.net_revenue, sp.sell_price,
(s.price_per_share* cast(s.total_quantity as int)) as cost_price,
(sp.sell_price - (s.price_per_share* cast(s.total_quantity as int))) as
total_profit
from IB_stock_sell_price sp join dma.client c on sp.corporateCo_id = c.ID
join dma.securities s on sp.security_id = s.security_id join
dma.corporateCo co on c.ID = co.client_id;
```

	corporateCo_id	name	net_revenue	sell_price	cost_price	total_profit
1	26	CustomMade	82000000	1530000	1500000	30000
2	27	Embedly	62000000	3075000	3000000	75000
3	28	Oyo	42000000	8040000	8000000	40000
4	29	Flipkart	22000000	985600	980000	5600
5	30	Boston Dynamics	20000000	1442400	1440000	2400





Visualisations created through Python using seaborn library.

4) Private Banking: The money this division brings in is simple. It is the sum of the total savings of the commoners who open an account with the bank. Of course, the net revenue or total profit would be from the high interest loan rates and low savings interest rates. This is captured in the revenue table, but for the sake of the simplicity of this project, this is not expanded.

```
select sum(total_savings) as total_inflow from dma.commoner;
```

	total_inflow
1	951009

5) The firm wants to look at the running total of the revenue of each of its division starting from 1st jan 2020 to till date to keep track of its progress. Below query calculates the running total ordered by month.

```
select div_id, month, revenue, sum(revenue)
over (partition by div_id order by month rows between unbounded preceding
and current row) as runningtotal
from dma.revenue;
```

	div_id	month	revenue	runningtotal
27	1	2022-03-01	3600000000	62100000000
28	1	2022-04-01	3700000000	65800000000
29	1	2022-05-01	3800000000	69600000000
30	1	2022-06-01	3900000000	73500000000
31	1	2022-07-01	4000000000	77500000000
32	1	2022-08-01	4100000000	81600000000
33	1	2022-09-01	4200000000	85800000000
34	1	2022-10-01	4300000000	90100000000
35	1	2022-11-01	4400000000	94500000000
36	1	2022-12-01	4500000000	99000000000
37	2	2020-01-01	500000000	500000000
38	2	2020-02-01	600000000	1100000000
39	2	2020-03-01	700000000	1800000000
40	2	2020-04-01	800000000	2600000000
41	2	2020-05-01	900000000	3500000000

6) Identify the total quantity of securities that can be sold now by the commoners i.e. release date should be before today's date. The purpose is to make arrangements for the sale.

```
select sp.security_id, sum(quantity) ready_to_sell_quantity
from dma.security_purchase sp
where DATEDIFF(day, CAST( GETDATE() AS Date ), release_date) > 0 group by
sp.security_id;
```

	security_id	ready_to_sell_quantity
1	100001	780
2	200002	325
3	300003	184
4	500005	184

7) What are the mean, min, max and range of salaries of employees in each division? T Display the name of the division and the number of employees in each division. he purpose of this query is for the HR department to compare its compensations with other firms to be able to hire best talent.

```
select d.NAME, avg(e.compensation) as mean_salary, min(e.compensation) as
min_salary, max(e.compensation) as max_salary, max(e.compensation) -
min(e.compensation) as range, count(*) as num_of_employees
from dma.employee e join dma.division d on e.div_id=d.ID
GROUP by d.NAME;
```

	NAME	mean_salary	min_salary	max_salary	range	num_of_employees
1	Asset Management	188527	115753	278131	162378	7
2	Investment Banking	209513	102294	300000	197706	8
3	Private and Corporate	208986	114220	298792	184572	8
4	Wealth Management	223477	120715	300000	179285	7

8) Identify how many employees work for each manager. Display the name of the manager, age, salary and the count. The purpose is to identify if the managers are being compensated properly or not.

```
with emp_count as (
select manager_id, count(*) as no_of_employees from dma.employee group by
manager_id except (select manager_id, count(*) from dma.employee
where manager_id is null group by manager_id))
select e.ID, e.name, e.compensation, c.no_of_employees from dma.employee
e join emp_count c on e.ID = c.manager_id;
```

	ID	name	compensation	no_of_employees
1	4	Venkat Krishnamurthy	300000	7
2	5	Bridgett Kovach	114220	7
3	7	Lexis Lofton	270000	6
4	10	Daniela Ceja	300000	6

9) Identify the commoner IDs of those commoners who availed all PCServices. The purpose of this analytical query is to list those commoners to provide them loyalty benefits.

```
select commoner_id from dma.PCprovision pc1 where
(select count(distinct pc_service_id)
from dma.PCprovision pc2
where pc1.commoner_id=pc2.commoner_id) =
(select distinct count(*) from dma.PCservice)
group by commoner_id;
```

	commoner_id
1	11
2	12

10) The firm has been saving its monthly revenue in the revenue's table. The data is incremental along the time series. The firm wants to predict what the revenues in the future would be for each of its division.

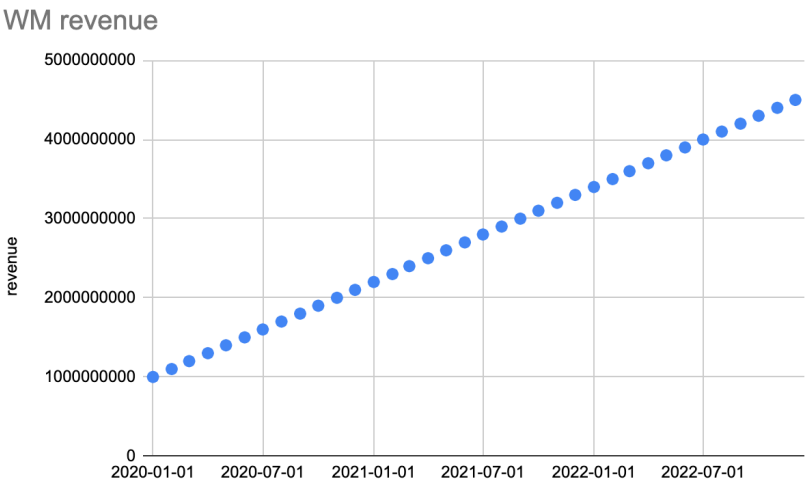
To begin with, we'll retrieve the revenue data from the database for division 1 I.e. wealth management and predict what the revenue would be in Jan 2023.

Initial data retrieval -

```
select * from dma.revenue where div_id=1;
```

	div_id	month	revenue
1	1	2020-01-01	1000000000
2	1	2020-02-01	1100000000
3	1	2020-03-01	1200000000
4	1	2020-04-01	1300000000
5	1	2020-05-01	1400000000
6	1	2020-06-01	1500000000
7	1	2020-07-01	1600000000
8	1	2020-08-01	1700000000
9	1	2020-09-01	1800000000
10	1	2020-10-01	1900000000

We get 36 rows - jan 2020 to dec 2022. Plotting revenue data as scatter plot-



Applying linear regression to predict future revenues-

```
#retrieving Wealth Management Revenue Data from database and saving in a dataframe.
wm_df = pd.DataFrame()
with pyodbc.connect(conn_string) as conn:
    with conn.cursor() as cursor:
        print('connection created')
        cursor.execute("select * from dma.revenue where div_id=1")
        row = cursor.fetchone()
        i=0
        while row:
            wm_df.loc[i,'date'] = str(row[1])
            wm_df.loc[i,'revenue'] = row[2]
            i+=1
            row = cursor.fetchone()
print(wm_df)
wmdf_temp = wm_df
```

```
[ ] #converting string type date to datetime and then to ordinal.
import datetime as dt
wmdf_temp['date'] = pd.to_datetime(wmdf_temp['date'])
wmdf_temp['date']=wmdf_temp['date'].map(dt.datetime.toordinal)
wmdf_temp
```

```
X = np.array(wmdf_temp['date']).reshape(-1, 1)
y = np.array(wmdf_temp['revenue']).reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
# Splitting the data into training and testing data

regr = LinearRegression()

regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
```

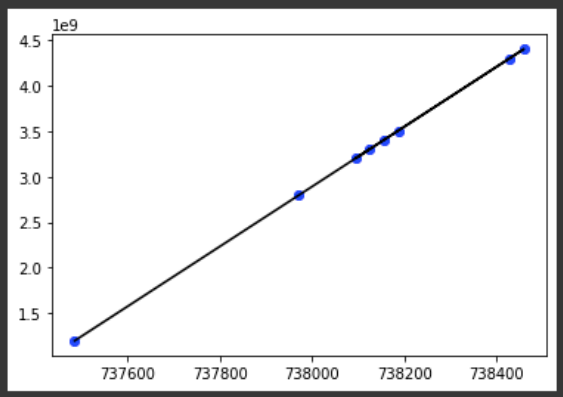
```
regr = LinearRegression()

regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
```

```
0.9999901766940232
```

```
y_pred = regr.predict(X_test)
plt.scatter(X_test, y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')

plt.show()
```




```
[ ] predicted_revenue = regr.predict([[pd.to_datetime('2023-01-01').toordinal()]])
    print(predicted_revenue[0,0])
```

```
4603430104.951172
```

NoSQL Queries and their analytical purpose

1) The firm wants to give certain free perks to common individual clients who have total savings in the bank between 10,000 USD and 30,000 USD. List the individual's details.

```
db.commoner.find({
    $and: [
 {"total_savings":{ $gt : 10000 }},
 {"total_savings":{ $lt : 30000 }}
 ]
});
```

commoner

_id	client_id	SSN	date_of_birth	total_savings	credit_score	age
10	10	956912164	2000-07-15T0C	24334	640	56
11	11	744813504	1960-09-13T0C	21243	578	41
17	17	774702760	1988-03-20T0C	25226	692	51
22	22	546807980	1997-02-16T0C	18377	574	26
23	23	851557351	1983-08-20T0C	17359	586	34

2) Find out how many employees are working under each manager. Do not list those employees who do have a manager.

```
db.employee.aggregate([
{ $match: {manager_id : {$exists: true, $ne: null}} },
{ $group: { _id: "$manager_id",
    num_of_employees:{ $count:{} }}}
]);
```

employee > num_of_employees

_id	num_of_employees
4	7
10	6
7	6
5	7

3) Finding the total revenue of each division over the months and sort in descending order. Analytical purpose: The firm wants to know which division is leading and the CEO would like to allocate more funding and resources to the divisions that can do better.

```
db.revenue.aggregate([
{
    $group: {
        _id: "$div_id",
        total_revenue: {
            $sum:"$revenue"
            }
        }
},
{
    $sort:{ "total_revenue" : -1}
}
]);
```

[Index]	total_revenue
0	162000000000
1	99000000000
2	81000000000
3	13500000000

4) Identify the cost price of each security. Cost price = cost price per share * quantity. Total cost price is calculated to get the amount the investment bank has to pay to the corporates after the securities have been sold.

```
db.securities.aggregate([{
    $project: {
        security_id: 1,
        total_cost_price: {$multiply: ["$price_per_share", "$total_quantity"]}
        }
}])
```

securities

_id	security_id	total_cost_price
100001	100001	1500000
200002	200002	3000000
300003	300003	8000000
400004	400004	980000
500005	500005	1440000

5) Get the details of the investment bankers who have underwritten till now. The firm wants to award the bankers for their service.

```
var underwriters = db.underwriting.distinct("IBanker_id")

db.employee.find({"ID": {"$in": underwriters}});
```

employee

ID	SSN	name	role	date_of_joining	compensation	house	street	state
1	889599561	Vishal Basutkar	Investment Bank	2021-09-10T0C	200000	1448	Washington Stre	MA
6	205602683	Keenan Lieberm	Investment Bank	2003-02-22T0C	112364	8167	Flatlands Avenue	MA

6) Identify how much each AM client has paid to the firm in terms of fees for the services and the profit share.

```
db.AMprovision.aggregate([
{
    $group: {
        _id: "$AMclient_id",
        total_profitshare: {$sum: "$profitshare"},
        total_fees: {$sum: "$fee"}
    }
},
{
    $project: {
        AMclient_id: 1,
        total_fee: "$fee",
        total_profitshare: "$profitshare",
        total_pay: {$add: ["$total_fees", "$total_profitshare"]}
    }
}]);
```

AMprovision

_id	total_pay
8	5055
27	6330
28	8496
7	7367
6	5877

Connecting database to Python Application

Back-end of applications generally written in high level programming languages like JAVA, Python, etc. can execute SQL queries by connecting the database using a connection string. Connection string has the server details and the connection credentials that can be retrieved from the Azure Portal. Generally, resources in cloud have firewall rules and hence even if we have the credentials and server details, the firewall may block the connectivity. We can to save the IP address range of the servers of the applications in the azure portal as firewall rules. Below are the screenshots of a python application connecting to the database in Google Colab and executing a simple query.

```
import pyodbc
server = 'dmagroup22.database.windows.net'
database = 'dmadb'
username = 'CloudSA169ab7fd'
password = '{DMAGroup22}'
driver= '{ODBC Driver 17 for SQL Server}'
conn_string = 'DRIVER='+driver+';SERVER=tcp:'+server+';PORT=1433;DATABASE='+database+';UID='+username+';PWD='+password
```

3) Find the amount of money brought in by the P&C division.

```
with pyodbc.connect('DRIVER='+driver+';SERVER=tcp:'+server+';PORT=1433;DATABASE='+database+';UID='+username+';PWD='+password) as conn:
    with conn.cursor() as cursor:
        cursor.execute("select sum(total_savings) as total_inflow from dma.commoner;")
        columns = [column[0] for column in cursor.description]
        print(columns)
        row = cursor.fetchone()
        while row:
            print (str(row[0]))
            row = cursor.fetchone()
```

```
['total_inflow']
951009
```

Conclusion

Through the Database Management for Analytics course taken by Prof. Venkat Krishnamurty, we understood how to take a business problem and create an efficient database system and apply analytics on the data. As a part of this project, we made a realistic project idea that would apply to all financial firms like JP Morgan, Bank of America, UBS, etc. that provide financial services like Wealth Management and Investment Banking. We understood the clients of each division and how each division generates revenue for the firm. Through the conceptual models EER and UML, we could identify the entity types and relationship types between them. We could specialise, generalise, categorise entity types, identify weak and strong entity types and create the most efficient models. Then we created the relational models from the EER diagram and normalised to 4th normal form.

Since most of the firms are migrating from their on-premise Oracle and MySQL databases to cloud, we implemented our solution in the cloud by exploring Microsoft Azure. Database creation, connection to the database through the GUI tool- Azure Data Studio was fairly simple and we could proceed with creating the tables with all the foreign key rules. The GUI tool also enabled us to create a notebook where we could write all our SQL queries for analytics.

Migrating data from Azure Database to MongoDB was done under a minute. The server details were required to be provided in Studio 3T, the GUI tool for MongoDB and after the connection was made, the tables were converted to documents. We could also perform simple analytical queries too.

Future scope of this simple database application is to expand the database across all divisions of the bank, connect the database to a visualisation tool like Tableau to create dashboards that could benefit the upper management of the firms to understand the summarised view of the firm. We can perform further analytics to target clients through network graphs based on current client data and bring in more clients to the firm.