COS20015 (Fundamentals of Database Management)

Merging both Research Report and the Database task together

(in a single PDF)

Name: SM Ragib Rezwan

ID: 103172423

Note: In this pdf, the first parts will focus on Research Report while the second part will focus on the database task. Both these will have their own titles, content list, etc , so do not be misled by them.

COS20015 (Fundamentals of Database Management)

Research Report: Is Relational database better than NoSQL database?

(using MariaDB and MongoDB to compare and contrast)

Name: SM Ragib Rezwan

ID: 103172423

CONTENT:

Abstract:

Databases are basically ways to store and access groups of data in an organized, efficient manner. Although there are several types of them currently existing in the market, they are all used to perform these two functions, albeit in slightly different ways (depending upon the needs of the company).

Among them, the most popular and well-known one is Relational Database. But NoSQL databases are also rising in demand in this current fast paced world. This arises the question: Is Relational database better than NoSQL database? Or has the time come to switch to NoSQL?

Introduction:

The dawn of the age of internet has made the world increasingly interconnected, leading to a high demand in storage, access, modification and deletion of data for both personnel and organizations. This has led to the development of a group of software, each of which can accomplish these tasks in different ways. This group of software is what we know as databases.

Currently in the world, different types of databases (like Centralized, Cloud, Object Oriented databases, etc *(Javatpoint, 2021)*) exists, each using different Query languages to help the users find and use the information they need effectively, reliably and efficiently. But even so most organizations generally tend to focus on two types: Relational and NoSQL databases.

Relational databases are the type of databases where structured information is stored following a rigid schema. Here, tables are made for each entity and relationships are made to connect those tables (and thus link the entities). Also, each table has columns which denote the attributes associated with that entity and rows to denote the records related to that entity. Furthermore, this type of databases either use SQL or a "dialect" of it (ie MYSQL) to search though its tables for specific data to access, update or delete and follows normalization principles (at least upto 3NF) to reduce data redundancy and increase data integrity.

On the otherhand, NoSQL databases are the types which store all forms of data (with main advantage laying in the use of semi structured and unstructured data), without enforcing any schema on the data stored. Thus, it can be realized that they don't follow normalization principle at all! Furthermore, although there are several types of such database (like Key-Value storage, Graph database, Wide-column Store, etc.) available, the only difference between them is in the slightly different way the data is stored and everything else is more or less the same.

So, although Relational and NOSQL databases are quite different from one another, we can see that they still basically perform the same functions as every other databases. Hence comes the obvious question: Is Relational database better than NoSQL database? Or has the time come to switch to NoSQL ones?

Method:

Although several Relational (like Oracle, PostgreSQL, SQL server, etc.) and NOSQL databases (like HBase, Cassandra, Haystack, etc.) exist, here I am going to only use MariaDB and MongoDB to compare them. This is because:

a) Both of them are very popular and most commonly used in the current world
b) Both of them are open source, enabling me to freely modify and use it without worry
c) Both of them have been used in varying degree in this course. So, I have a good idea on what they can and can't do, and thus, know most of the external factors that I need to control (to ensure proper comparison can be made between them).

To ensure that the comparison is fair, I am going to use the database I previously made in MariaDB (for my "custom database" task) and utilize its data to make the databases in MongoDB. This ensures that both databases have the same data stored in them and so any differences present will be due to the database type used to store them.

[Note: if you wish to see the database setup and full details for MariaDB, please see the pdf for the custom database task]

[Note: if you wish to see the database setup and full details for MongoDB, please see between conclusion and reference]

i) Brief information regarding the database made in MariaDB:

Here I used MYSQL query language to make a database called companynew which had tables for: Product_details, subsidiary_company_details, shipments, Orders, Customers_details, Employee_details, Individual_Employee_position, Employee_position, Employee_and_Branch_details and Branch_details.

In Product_details table, there were attributes for product_id, product_name, product_type, product_material, product_price, product_comment, subsidiary_company_id, product_purchase_date_from_subsidiary,product_in_stock. The table also had 5 rows of data in it, with primary key product id and foreign key subsidiary_company_id.

In subsidiary_company_details table, there were attributes for sunsidiary_company_id, subsidiary_company_name, subsidiary_company_address, subsidiary_company_city,subsidiary_company_postcode, subsidiary_company_phonenumber. The table also had 4 rows of data in it with primary key subsidiary_company_id

In shipment table, there were attributes for shipment_id, order_id,shipment_date . The table also had 4 rows of data in it, with primary key shipment_id and foreign key order_id.

In orders table, there were attributes for order_id, customer_id, product_id, employee_id, order_stock, order_date, order_comment. The table also had 4 rows of data in it with primary key order_id and foreign key customer_id, product_id and employee_id. Also, it is a weak entity table

In customers_details table, there were attributes for customer_id, customer_fname, customer_lname, customer_gender, customer_address, customer_city, customer_postcode, customer_phone . The table also had rows of data in it with primary key customer_id.

In Employee_details table, there were attributes for mployee_id, employee_fname, employee_lname, employee_gender, employee_date_of_birth, employee_address, employee_city, employee_postcode, employee_phone, employee_current_salary, employee_individual_comment. The table also had 5 rows of data in it with primary key employee_id.

In Individual_Employee_position table, there were attributes for employee_id, employee_position . The table also had 5 rows of data in it with foreign key employee_id, employee_position. Also, it is a weak table.

In Employee_position table, there were attributes for employee_position and employee_position_comment . The table also had 3 rows of data in it with primary key employee_position.

In Employee_and_Branch_details table, there were attributes for branch_id, employee_id . The table also had 5 rows of data in it with foreign key branch_id, employee_id. Also, it is a weak table

In Branch_details table, there were attributes for branch_id, branch_name, branch_comment . The table also had 2 rows of data in it with primary branch_id.

Furthermore I have also made 3 views in it for full_employee_details, full_order_details and full_product_details with necessary attributes (via de-normalization) to enable users to easily find all important the data in just 3 places, instead of searching through all the tables present in the database and wasting time.

ii) <u>Brief information regarding the database made in MongoDB:</u>

[Note: MongoDB is a document database which is basically a subclass of the key-value store database of NoSQL and follows JSON format]

Here I used NOSQL to make a database called newcompanydb and a collection in it called newCompany. Inside that collection, I made 3 documents for branch, subsidiary company and customer details.

In Branch, I made attribute/keys: ID, name, comment for branches and an array called Employee which contained ID, name, gender, date of birth, address (another array inside it that had address, city and postcode), phone, salary, position, comment for all employees in that branch.

In Subsidiary Company, I made attribute/keys: ID, name, city, phone for subsidiary companies and an array called Products which contained ID, name, type, material, price, comment, purchase_date, stock for all products supplied by that subsidiary company.

In customer details, I made attribute/keys: ID, name, gender, address, city, postcode, phone for customers and an array called orders which contained order ID, product ID, Product name, Ordered Product stock, order product cost, employee ID (of one who sold the product), employee name(of one who sold the product) , order_date, shipment_ID, shipdate, Order comment for all orders made by those customers

Here, I am going to compare them on the following matrixes: Data structures and retrieval, concurrency model/isolation levels, speed of access, error detection in update and database security. For each of them I will do both brief literary comparison and code implementations in the results. Then I will provide further details and analysis about each in the discussions.

Results:

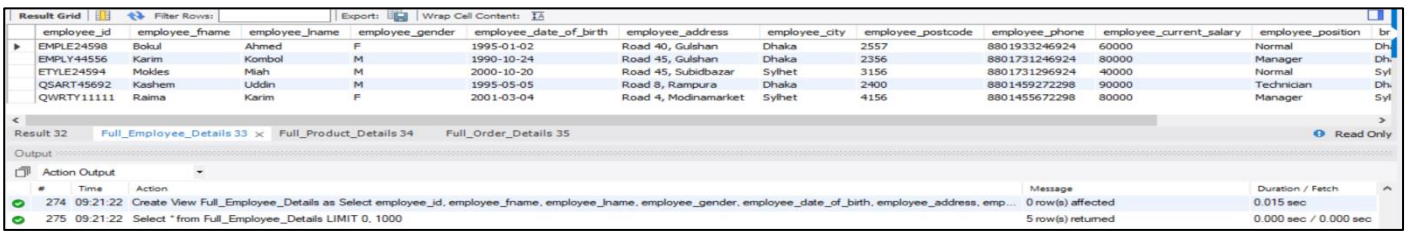
a) Data structures and retrieval:

[Note: Data structure" is the way used to provide an efficient method of storing and organizing groups of data elements in "temporary memory", so that they can be easily modified and accessed. This is different from "database structure" which manages data stored in "permanent memory" (Lithmee, 2019)]

In the MariaDB, MySQL provides good data structure and retrieval by utilizing techniques like denormalization via views and by using indexes (like primary, B+ Tree, and Hash).

Command and output for a sample view:

```
Create View Full_Employee_Details as
Select employee_id, employee_fname, employee_lname, employee_gender, employee_date_of_birth,
employee_address,employee_city, employee_postcode, employee_phone, employee_current_salary,
employee_position,branch_name, employee_individual_comment

 From employee_details e Natural Join employee_and_branch_details eb Natural Join branch_details b
NATURAL Join individual_employee_position ie NATURAL Join employee_position
Order by employee_id;

Select *
from Full  Employee  Details;
```



employee_id	employee_fname	employee_lname	employee_gender	employee_date_of_birth	employee_address	employee_city	employee_postcode	employee_phone	employee_current_salary	employee_position	br
EMPLE24598	Bokul	Ahmed	F	1995-01-02	Road 40, Gulshan	Dhaka	2557	8801933246924	60000	Normal	Dh
EMPLY44556	Karim	Kombol	M	1990-10-24	Road 45, Gulshan	Dhaka	2356	8801731246924	80000	Manager	Dh
ETYLE24594	Mokles	Miah	M	2000-10-20	Road 45, Subidbazar	Sylhet	3156	8801731296924	40000	Normal	Syl
QSART45692	Kashem	Uddin	M	1995-05-05	Road 8, Rampura	Dhaka	2400	8801459272298	90000	Technician	Dh
QWRTY11111	Raima	Karim	F	2001-03-04	Road 4, Modinamarket	Sylhet	4156	8801455672298	80000	Manager	Syl

Result 32 | Full_Employee_Details 33 | Full_Product_Details 34 | Full_Order_Details 35 | Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
274	09:21:22	Create View Full_Employee_Details as Select employee_id, employee_fname, employee_lname, employee_gender, employee_date_of_birth, employee_address, emp...	0 row(s) affected	0.015 sec
275	09:21:22	Select * from Full_Employee_Details LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

On the other hand, in MongoDB, all the objects and attributes related to the entity are kept in a single document. So, only need to query that document normally to retrieve the data, instead of implementing any extra techniques.

Command used:

```
db.newCompany.find().pretty()
```

Output:

```
> db.newCompany.find().pretty()
{
        "_id" : ObjectId("6185bfdf859dfb5e68d78bd4"),
        "Branch" : [
                {
                        "ID" : "BRNCH40002",
                        "Name" : "Dhaka Branch",
                        "Comment" : "Exclusive Branch in Dhaka",
                        "Employee" : [
                                {
                                        "ID" : "EMPLE24598",
                                        "Name" : "Bokul Ahmed",
                                        "Gender" : "F",
                                        "Date_of_birth" : "1995-01-02",
                                        "Address" : {
                                                "address" : "Road 40, Gulshan",
                                                "city" : "Dhaka",
                                                "postcode" : "2557"
                                        },
                                        "Phone" : "8801933246924",
                                        "Salary" : "60000",
                                        "Position" : "Normal",
                                        "Comment" : "Clumsy Employee"
                                },
                                {
                                        "ID" : "EMPLY44556",
                                        "Name" : "KarimKombol",
                                        "Gender" : "M",
                                        "Date_of_birth" : "1990-10-24",
                                        "Address" : {
                                                "address" : "Road 45, Gulshan",
                                                "city" : "Dhaka",
                                                "postcode" : "2356"
                                        },
                                        "Phone" : "8801731246924",
                                        "Salary" : "80000",
                                        "Position" : "Manager",
                                        "Comment" : "good old employee"
                                },
```

b) <u>concurrency model/isolation levels:</u>

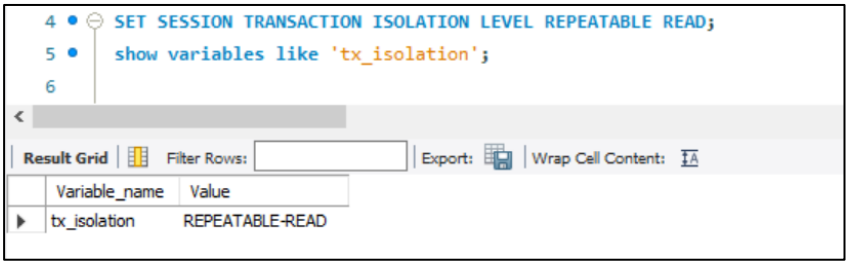
[Note: Data Concurrency is the ability to let many users work on the same data at the same time, (keeping it upto date) which increases the risks of errors (like Lost update, Dirty reads, non-repeatable reads, phantoms, etc.) when proper precautions are not maintained]

In MySQL, this is maintained by using isolation levels: Read Uncommitted, Read Committed, Repeatable Read, and Serializable; with the default being REPEATABLE READ. This is usually altered by using "SET SESSION TRANSACTION ISOLATION LEVEL <<Isolation level name>>" Command before performing the transaction.

Command used:

```
SET SESSION TRANSACTION ISOLATION LEVEL
 REPEATABLE READ       //setting to repeatable read
Show variable like 'tx_isolation'          //wrote this so that the change can be observed
```

Output:



In NOSQL, it is maintained by using ReadConcern and Write concern, where read concern can be set as local, available, majority, linearizable and snapshot (with default being local) and write concern can be set as majority, 0, 1, 2 (or any other number), or as a custom name (with default being majority). The read conditions for transaction can be set by adding readconcern and level after the desired query (eg "db.collection.find().readConcern(<level>)"command) and write concern can be set using the "db.setWriteConcern(<level>)" command

[Note: For defaults for read and write concern, I had to look up on website online, as in the MongoDB I used, there weren't any default values given for them from before. The website I used has been referred to in the discussion part for this area]

Command used:

db.getName()	show name of database used
db.setWriteConcern(1)	set the value to 1 which means it will acknowledge from nary before performing the task
db.getWriteConcern()	show that it has been set

Output:

```
> db.getName()
newCompanydb
> db.setWriteConcern(1)
> db.getWriteConcern()
WriteConcern({ "w" : 1 })
>
```

c) <u>Speed of access :</u>

In order to measure speed of access, I ran a query on both databases and noted the time it took for them to run it. Since I want both queries output the same result, I tried to find "all employee details, including their branch details" using the query.

[Note: set profile and show profile are used here to track and measure time taken to run the query]

For MariaDB, it can be done in two ways:

1) commands used are (without using technique like view):

```
set profiling=1;
Select *
 From employee_details e Natural Join employee_and_branch_details eb Natural Join branch_details b
Order by employee_id;
show profiles;
```

0.00276370	Select * From employee_details e Natural Join ...
0.00101010	Select * From employee_details e Natural Join ...
0.00166770	Select * From employee_details e Natural Join ...
0.00119120	Select * From employee_details e Natural Join ...

And the times it took:

So, on average it took: **0.00165818 second (1.65818 milliseconds)**

Name: SM RAGIB REZWAN ID: 103172423

2) commands used are (using view):

```
set profiling=1

Select *

from Full_Employee_Details

Order by employee_id;

show profiles;
```

And the time it took:

0.00172020	Select * from Full_Employe...
0.00167810	Select * from Full_Employe...
0.00162720	Select * from Full_Employe...
0.00094030	Select * from Full_Employe...

So, on average it took:

0.00149145 second

(1. 49145 milliseconds)

For MongoDB, commands used are:

```
db.setProfilingLevel(2)

db.newCompany.find( { "_id" : ObjectId("6177c86fd5d09d331b8a6198") }).pretty().explain("executionStats")
```

And the output is:


```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 1,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 1,
        "totalDocsExamined" : 1,
        "executionStages" : {
                "stage" : "IDHACK",
                "nReturned" : 1,
                "executionTimeMillisEstimate" : 0,
                "works" : 2,
                "advanced" : 1,
                "needTime" : 0,
                "needYield" : 0,
                "saveState" : 0,
                "restoreState" : 0,
                "isEOF" : 1,
                "keysExamined" : 1,
                "docsExamined" : 1
        }
```


This output stated that the time taken was 0 milliseconds. This seemed unusual to me and thus I ran it several times, while controlling all external factors (ie keeping all other applications off, checking just after laptop had boot up, etc). Even so, it still kept coming 0 milliseconds. So had to consider this data as accurate, noting the time as either 0ms, or very, very close to that.

d) Error Detection in Update:

Attempting to Update without setting any restriction in MariaDB:

Command:

```
Update employee_details
set employee_fname = "Bola";
```

Output:

```
2  05:53:02  Update employee_details set employee_fname = "Bola"  Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY ...  0.000 sec
```

Thus we can see that MariaDB automatically stops such an error as it is in safe update mode by default.

But in Mongodb, we can do the same thing by using the command:

```
db.newCompany.updateOne(   { _id: ObjectId( "6177c86fd5d09d331b8a6198")},
  {    $set: { Branch:{"Employee.Name" :"Bola Ahmed" }}    } )
```

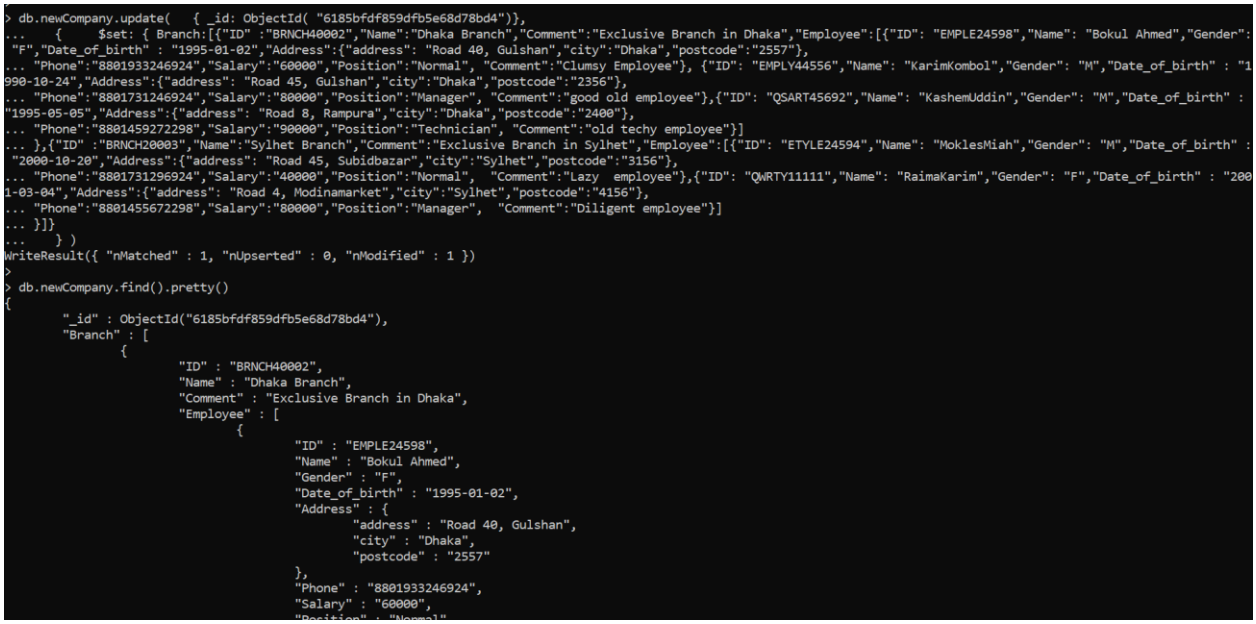
But this time it accepts the update command, giving following output:

```
> db.newCompany.updateOne(    { _id: ObjectId( "6177c86fd5d09d331b8a6198")},    {     $set: { Branch:{"Employee.Name" :"Bola Ahmed" }}    } )
{ "acknowledged" : true, "matchedCount" : 0, "modifiedCount" : 0 }
> db.newCompany.updateOne(    { _id: ObjectId( "6185bfdf859dfb5e68d78bd4")},
...    {     $set: { Branch:{"Employee.Name" :"Bola Ahmed" }}
...    } )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.newCompany.find().pretty()
{
        "_id" : ObjectId("6185bfdf859dfb5e68d78bd4"),
        "Branch" : {
                "Employee.Name" : "Bola Ahmed"
        }
}
{
        "_id" : ObjectId("6185bfdf859dfb5e68d78bd5"),
        "Subsidiary Company" : [
                {
                        "ID" : "SCOMP00001",
                        "Name" : "Office Products",
                        "City" : "Dhaka",
                        "Phone" : "8801972448031",
                        "Product" : [
```

Thus, to fix the change we made in Mongo, we will need to use the setup data from before to write following command:

```
db.newCompany.explain("executionStats").update(   { _id: ObjectId( "6185bfdf859dfb5e68d78bd4")},
  {    $set: { Branch:
[
{"ID" :"BRNCH40002","Name":"Dhaka Branch","Comment":"Exclusive Branch in Dhaka","Employee":
[{"ID": "EMPLE24598","Name": "Bokul Ahmed","Gender": "F","Date_of_birth" : "1995-01-02","Address":{"address": "Road
40, Gulshan","city":"Dhaka","postcode":"2557"},"Phone":"8801933246924","Salary":"60000","Position":"Normal",
"Comment":"Clumsy Employee"},
{"ID": "EMPLY44556","Name": "KarimKombol","Gender": "M","Date_of_birth" : "1990-10-24","Address":{"address": "Road
45, Gulshan","city":"Dhaka","postcode":"2356"},"Phone":"8801731246924","Salary":"80000","Position":"Manager",
"Comment":"good old employee"},
{"ID": "QSART45692","Name": "KashemUddin","Gender": "M","Date_of_birth" : "1995-05-05","Address":{"address":
"Road 8,
Rampura","city":"Dhaka","postcode":"2400"},"Phone":"8801459272298","Salary":"90000","Position":"Technician",
"Comment":"old techy employee"}]
},
{"ID" :"BRNCH20003","Name":"Sylhet Branch","Comment":"Exclusive Branch in Sylhet","Employee":
[{"ID": "ETYLE24594","Name": "MoklesMiah","Gender": "M","Date_of_birth" : "2000-10-20","Address":{"address": "Road
45, Subidbazar","city":"Sylhet","postcode":"3156"},"Phone":"8801731296924","Salary":"40000","Position":"Normal",
"Comment":"Lazy  employee"},
{"ID": "QWRTY11111","Name": "RaimaKarim","Gender": "F","Date_of_birth" : "2001-03-04","Address":{"address": "Road
4, Modinamarket","city":"Sylhet","postcode":"4156"},"Phone":"8801455672298","Salary":"80000","Position":"Manager",
"Comment":"Diligent employee"}]}
]
} } )
```

Output:



e) Database security:

[Note: Database Security is the use tools or methods in a database to ensure security of the data in it]

In MariaBD, it is done by End-to-end encryption, by Firewall and data masking, by Auditing, by Denial of service protection and by Pluggable authentication and role/group authorization. Each of these performs in different ways, providing protection against various concerns in the database. For instance, End to end encryption is done using AES and TLS where AES is used to encrypt the data being

```
Use test;

create table t

(name_stuff blob not null);



INSERT INTO t (name_stuff) VALUES
(AES_ENCRYPT('Bigar',SHA2('password',512)));

SELECT name_stuff, AES_DECRYPT(name_stuff,
SHA2('password',512) )

FROM t;
```

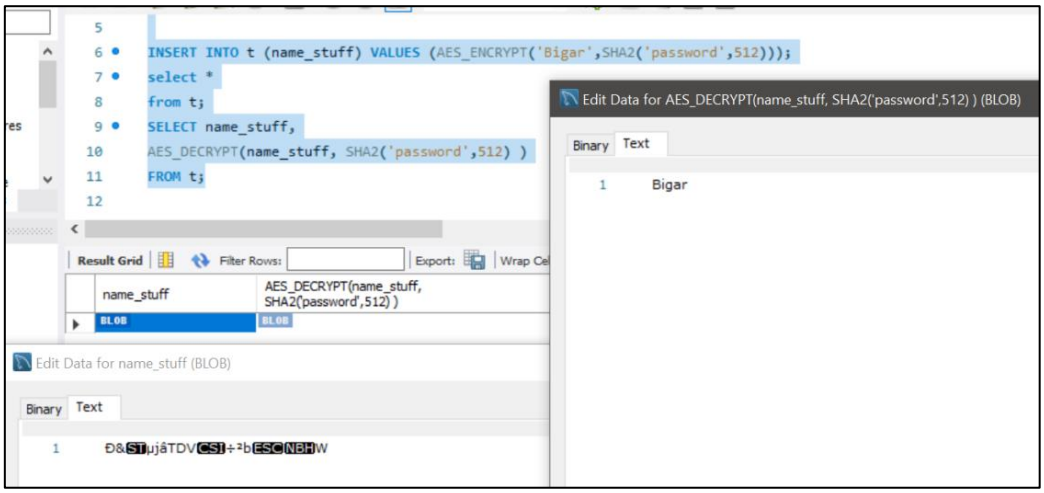
//here it encrypts the string “Bigar” inserted into the table using the key key_string “SHA2” and stores a binary string in the field.

//Furthermore, the only condition here is that the datatype of the field must be BLOB.

//Here I wrote two columns in the query, one to show how it looked encrypted and one to show what the actual data was

[In practical case, just directly use AES_DECRYPT part only and ignore name_stuff part]
[Note: If you want to use VARCHAR, rather than BLOB, then convert the encrypted binary to Base64]

Outputs:



On the other hand, in MongoDB, it is done by encryption, authentication, authorization, auditing. Like in case of MariaDB, each these focus on providing protection against different threats to the database. For instance, for encryption MongoDB uses Encrypted storage engine which ensures data is encrypted both on network and on disk. It can be set using the following command:

```
clientEncryption = db.getMongo().getClientEncryption();
 "phone" : clientEncryption.encrypt(
   UUID("64e2d87d-f168-493c-bbdf-a394535a2cb9"),
   "123-45-6789",
    "AEAD_AES_256_CBC_HMAC_SHA_512-Deterministic")
```

//getting the Client encryption, storing it in variable, then encrypt by using a key (the UUID), value you want to encrypt (ie the phone number) and an algorithm you wish to use to encrypt it.

[Note: Unfortunately, this can only be used in mongodb's enterprise mode (not in the community mode). Thus I cant show the output for it. But basically you will have to just set up a strict schema saying what sort of encryption the objects will have and then run the commands given to insert the data using the encryption]

Discussion:

- In data structures and retrieval, MariaDB has to use techniques like view and indexing to maintain it. Although denormalizing using views sounds counter intuitive with respect to its "relational database structure", using views actually helps saves time and resources for big queries. That's because it creates a pre-optimized table that contains data of multiple tables stored in it, avoiding the use of the expensive joins to query and update data. Furthermore, with the help of indexing (which is basically a look up table), pointers can be assigned to

values on one or more columns (like on only primary key for primary index, any attribute for B+ tree, attributes that are highly selective for Hash maps) making it more efficient for databases as they only have to see the starting row at each block to find where their desired data is located (instead of checking all the rows in all blocks).

On the other hand, in MongoDB's case, it is a document database where related entities are stored in a collection using JSON format (attribute stored as name-value pair separated by colon, different attributes separated by commas, and objects/sub-documents related to that entity) separated using curly brackets). Thus, (as seen in the results) we don't have to use any extra techniques (like creating and maintaining views or anything else) as everything related to an entity are kept in a single document, instead of being separated into multiple tables. So only need to query that to retrieve the data, which is very efficient and also more readable (as the name beside the value directly tells you what it represents).

Thus, regarding to data structure and retrieval, it is better to use MongoDB instead of MariaDB as you can find all the needed data in one place (without using any extra techniques) in a readable manner from the getgo, instead of looking through and linking multiple related tables.

- In terms of Concurrency, MariaDB does this by maintaining isolation levels, Read Uncommitted, Read Committed, Repeatable Read, and Serializable, with read uncommitted having highest performance and serializable having the highest isolation level. Read committed stops no anomaly what so ever and is the worst one to be used here. Next level is Read Committed which stops lost update (update done to data in a transaction lost as it was overwritten by another at the same time) and dirty reads (transaction reading a data that has been modified but not committed by another transaction). After this comes Repeatable Read which stops lost update, dirty read and also non-repeatable read (transaction reads a data without committing but finds different value when it re-reads the same data). At the very last comes Serializable which basically stops all anomalies (including phantoms where rerunning the same query turns up new rows) but needs to wait for each transaction to run one at a time, which makes it slow.

  In MongoDB, it is maintained a bit more flexibly, allowing people to alter read concern and write concern to set different isolation levels of reading and writing the data. According to the MONGODB Manual, by default, read concern is set as local (query instantly returns data without guarantee whether majority of replica set received it or not, available for use with or without causally consistent sessions and transactions) and write concerns is set as majority (need acknowledgement from majority of the primary and secondary present before writing to it).

  [Note: Causal consistency means Operation that logically depends on a preceding operation]

  Furthermore, for read concern, they can also be set as available (same as local, but only used without causally consistent session and transaction), majority (returns data that majority of replica set acknowledged), linearizable (returns data that majority-acknowledged write has successfully completed, only used without causally consistent session and transaction) and snapshot (read data after returned from transaction complete with write concern majority).

Moreover, for write concern, they can also be set as 0, 1, 2 or any other number (depends on how many primary and secondary Requests must be acknowledged for the write operation) or as custom name (needs acknowledgement from tagged members before writing).

Although, we can see MongoDB has good flexibility and control in terms of reading and writing data, it doesn't actually support transactions. Instead, it offers transaction-like semantics. So, it is possible for application to return an intermediate data during commit or rollback, which can lead to dirty read scenario (MongoDB, 2021). Thus, in this case, using MariaDB is better than using MongoDB.

- In terms of speed of access, we can see that although both databases are very fast (as they take milliseconds on average to run the queries outputting the same data for each), MongoDB runs the query the fastest (as it takes 0ms or a time very close to it on average). That's mainly because in MongoDB, it just needs to load up a single document while in MariaDB, it needs to either join multiple tables or run a pre-optimized view. Thus, in terms of speed, MongoDB is far, far ahead of MariaDB and thus for high speed performance, it is better to use MongoDB rather than MariaDB.

- In terms of detecting errors in update, in MariaDB, we can see that by default it has a safe update mode which prevents us from updating a table without setting any restriction to the update. This is quite beneficial as it prevents the scenario where "all the rows for an attributes in a table have been mistakenly updated to the same value" (as we didn't notice that we had forgotten to set a restriction). Otherwise, it would end up being a big issue as then we would have to spend a long time looking through the back up files to fix the data back to their correct values.

  But in MongoDB, there is no such safety feature. Instead, as it follows no specific schema, it believes that the changes made were intentional and thus acts accordingly (as seen in the output in that section where it even removed all the other attributes!). Thus, in the end, we would have to reupdate all the data by relying on backup or setup data to fix the error, which would take a long time (and also may not be accurate)

  Although this might seem like a mundane error that a professional database expert won't ever make, considering the huge responsibility database experts have nowadays (Maintaining software updates, data security, authentication, capacity monitoring, performance tracking, specialized data handling ,etc. (Jon Cowling , 2016)), such slipups are actually quite common. Thus, it is better to use MariaDB as it has this default safety feature built into it and hence prevent the entire database from being kept offline for a longtime just fix the error cause by this slight slipup.

  [Note: the links for websites for mariadb and mongodb data security features are given in reference]

- In terms of database security, MariaDB does it in 5 ways: End-to-end encryption, Firewall and data masking, Auditing, Denial of service protection and Pluggable authentication and role/group authorization. End-to-end encryption encrypts the data passed using AES(Advanced Encryption Standard) and also the connection itself using TLS(Transport Layer Security) (as seen in the commands before). Firewall and data masking is used to prevent malicious attacks like SQL injection and to hide sensitive data by intercepting and blocking certain queries (implemented by making a masking filter using Maxscale configuration and a json file with masking rules detailed in it). Auditing tracks everything on the database (all queries, tables operations, etc) including when and who made the change (need to install Mariadb audit plugin, activate it and choose which events to audit like connection, queries, tables, etc). Denial of service protection mainly prevents Dos and DDos attacks (implemented by configuring a result limiting filter to block queries that intend to slow down database by returning huge results). Pluggable authentication and role/group authorization ensure that only authenticated and authorized personnel can access and change the data in the database (implemented using LDAP authentication, one-time passwords, two factor authentication, setting up user accounts and its password,etc)

- On the other hand, in MongoDB does it by 4 ways: Encryption, Authentication, Authorization, and Auditing. Encryption is done using MongoDB's Encrypted storage engine which ensures data is encrypted both on network and on disk. Authentication is done by integration LDAP, Kerberos, etc external security to ensure person has the needed authentication to access the data. Authorization is done by providing different roles to user for the application, limiting their access to only that which is needed. Auditing is done by using a native log that keeps tracks of all operations and access performed on the database.

  [Note: I have been unable to set the encryption on MongoDB as I don't have access to MongoDB's Enterprise Mode and thus stated what would be seen and the way to implement it, following the MongoDB manual]

  Now, although these two have similar security, we can see that there is one feature that separated both of them. In MariaDB, there are features present to prevent DDOS attacks which is absent in MongoDB. Thus considering the current situation, where DDoS attack has been on the rise (as seen by David Warburton in F5 labs where he noticed that it has increased by 55% between Jan 2020 to March 2021), MariaDB has an easy win compared to MongoDB.

Conclusion:

So, by observing the two databases made for the same data in the same company, we have found different benefits and drawbacks the two databases. Thus, by weighing and comparing them, we can come to the conclusion that for the database made, it is better to make it using MariaDB instead of MongoDB. This is because although MongoDB has several advantages, like having extremely fast query speed, benefit of being able to use any data type, no rigid schema, etc., it has serious drawbacks, like not having reliable transaction and isolation levels, not having a safety mode for

update, not having any security setup to prevent ddos attack, etc. Furthermore its biggest advantage of being able to use unstructured data becomes irrelevant here as the data (stored by company) are in structured form. Hence, MariaDB, with its strict schema, proper relation between tables, extra optimization techniques for performance, reliable transaction and isolation levels, better data security,etc. prevails over MongoDB here.

But does it mean MariaDB will always be better than MongoDB? Not really.

That's because:

a) When we compared MariaDB and MongoDB, only data allocated to one company's (ie my imaginary "furnitures international" company) use was noted, which lead to a certain format for MongoDB's setup (ie 3 documents containing everything important). But different formats (like making individual documents for each person or product, keeping data in different collections in the database, etc) could have also been written for it! Thus, since not all possible formats have been tested, there isn't a 100% guarantee that the comparisons written here will also be accurate for those Scenarios as well.

b) Furthermore, as seen before, the encryption command need MongoDB's enterprise version to run the command. This showed that MongoDB's Enterprise had more commands, which were unavailable to those accessing it using MongoDB's Community version. Thus, it is possible that there were other commands present there which may had given MongoDB a better advantage against MariaDB if used here.

So, in reality, determining which database is better is more like "using the right tool for the right job". After all, if a company used only structured data, demands good transaction and database consistency, no redundancy, robust database, high database security, etc. then MariaDB is better as it is well suited for that situation. But if they need to deal with unstructured or semi-structured data, have fast access from getgo (without relying on extra techniques), flexible database schema (as need to store different item with different attributes), etc., then MongoDB is better instead.

So the current conclusion for "whether relational or NOSQL database is better" is: uncertain

That's because:

a) As stated before, it depends which company is using it and what it is using it for
b) Moreover, here we only compared among two databases (MariaDB and MongoDB) of those two types (Relational and NoSQL databases) instead of all databases that follow these two types.

So, It will be better to repeat this experiment firstly on different ways the database could also be made in MongoDB Enterprise (to have full access to all commands) and then on other Relational (like Oracle, PostgreSQL, SQL server, etc.) and NOSQL databases (like HBase, Cassandra, Haystack, etc.) That also exists. It is only then that we can make an accurate, valid, clear-cut answer to the question:

"Is Relational databases better than NoSQL databases?"

MongoDB database setup:

```
use newcompanydb
db.newCompany.insertMany(
[
	{"Branch":
		[
		{
	"ID" :"BRNCH40002",
	"Name":"Dhaka Branch",
	"Comment":"Exclusive Branch in Dhaka",
	"Employee":
		[
		{
			"ID": "EMPLE24598",
			"Name": "Bokul Ahmed",
			"Gender": "F",
			"Date_of_birth" : "1995-01-02",
			"Address":
				{
					"address": "Road 40, Gulshan",
					"city":"Dhaka",
					"postcode":"2557"
				},

			"Phone":"8801933246924",
			"Salary":"60000",
			"Position":"Normal",
			"Comment":"Clumsy Employee"
		},
		{
			"ID": "EMPLY44556",
			"Name": "KarimKombol",
			"Gender": "M",
			"Date_of_birth" : "1990-10-24",
			"Address":
				{
					"address": "Road 45, Gulshan",
					"city":"Dhaka",
					"postcode":"2356"
				},
```

```
                "Phone":"8801731246924",
                "Salary":"80000",
                "Position":"Manager",
                "Comment":"good old employee"
        },


        {
                "ID": "QSART45692",
                "Name": "Kashem         Uddin",
                "Gender": "M",
                "Date_of_birth" : "1995-05-05",
                "Address":
                        {
                                "address": "Road 8, Rampura",
                                "city":"Dhaka",
                                "postcode":"2400"
                        },

                "Phone":"8801459272298",
                "Salary":"90000",
                "Position":"Technician",
                "Comment":"old techy employee"
        }
        ]

},
{
"ID" :"BRNCH20003",
"Name":"Sylhet Branch",
"Comment":"Exclusive Branch in Sylhet",
"Employee":
        [{
                "ID": "ETYLE24594",
                "Name": "Mokles         Miah",
                "Gender": "M",
                "Date_of_birth" : "2000-10-20",
                "Address":
                        {
                                "address": "Road 45, Subidbazar",
                                "city":"Sylhet",
                                "postcode":"3156"
                        },

                "Phone":"8801731296924",
```

```
                "Salary":"40000",
                "Position":"Normal",
                "Comment":"Lazy  employee"
        },
        {
                "ID": "QWRTY11111",
                "Name": "Raima           Karim",
                "Gender": "F",
                "Date_of_birth" : "2001-03-04",
                "Address":
                        {
                                "address": "Road 4, Modinamarket",
                                "city":"Sylhet",
                                "postcode":"4156"
                        },

                "Phone":"8801455672298",
                "Salary":"80000",
                "Position":"Manager",
                "Comment":"Diligent employee"
        }]

}

        ]
},
{
"Subsidiary Company":
        [
        {
"ID": "SCOMP00001",
"Name":"Office Products",
"City":"Dhaka",
"Phone":"8801972448031",
"Product":
        [{
                "ID": "PHGTA00144",
                "Name": "Office table",
                "Type": "table",
                "Material": "wood",
                "Price": "6000",
                "Comment": "good office table made of wood",
                "Purchase_Date":"2019-02-16",
                "Stock":"40"
        },
```

```
        {
                "ID": "PRDTA00103",
                "Name": "Office chair",
                "Type": "chair",
                "Material": "wood",
                "Price": "4000",
                "Comment": "good office chair made of wood",
                "Purchase_Date":"2020-12-12",
                "Stock":"70"
        }]

},
{
"ID": "SCOMP00051",
"Name":"RFL",
"City":"Dhaka",
"Phone":"8801745336711",
"Product":
        {
                "ID": "PRDTA00140",
                "Name": "RFL sofa",
                "Type": "sofa",
                "Material": "plastic",
                "Price": "500",
                "Comment": "good RFL chair made of plastic",
                "Purchase_Date":"2018-10-12",
                "Stock":"90"
        }

},
{
"ID": "SCOMP00444",
"Name":"Stylish Products",
"City":"Dhaka",
"Phone":"8801973448031",
"Product":
        {

                "ID": "PRDTG00210",
                "Name": "stylish chair",
                "Type": "chair",
                "Material": "wood",
                "Price": "10000",
                "Comment": "good chair made of wood ",
                "Purchase_Date":"2020-12-15",
```

```
                    "Stock":"50"
            }

    },
    {
    "ID": "SCOMP05001",
    "Name":"Glassify",
    "City":"Sylhet",
    "Phone":"8801745336700",
    "Product":
            {
                    "ID": "PROOA00103",
                    "Name": "clear wardrobe",
                    "Type": "wardrobe",
                    "Material": "glass",
                    "Price": "8000",
                    "Comment": "good wardrobe made of glass       ",
                    "Purchase_Date":"2020-10-10",
                    "Stock":"90"
            }

    }
            ]
    },

    {
    "Customer Details":
    [
    {
            "ID": "ADBCA00554",
            "Name": "Kawsar          Hossain",
            "Gender": "M",
            "Address": "Road 10, Bonani",
            "City": "Dhaka",
            "Postcode": "4156",
            "Phone":"8801796958971"

    },
    {
            "ID": "ALBCA11020",
            "Name": "Anne Stuward",
            "Gender": "F",
            "Address": "Road 2, Rampura",
            "City": "Dhaka",
            "Postcode": "6915",
```

```
        "Phone":"8801692433190"

    },
    {
        "ID": "FABCA09548",
        "Name": "Robert          Brown",
        "Gender": "M",
        "Address": "Road 10, Modina",
        "City": "Sylhet",
        "Postcode": "3114",
        "Phone":"8801892884308",
        "Order":

        {
        "Order ID" :"ORDRE44990",
        "Product ID" :"PROOA00103",
        "Product name" :"clear wardrobe",
        "Ordered product stock" : "10",
        "Order product cost" : "80000",
        "Employee ID" : "ETYLE24594",
        "Employee name" :"Mokles       Miah",
        "Order date" : "2021-06-26",
        "Shipment_ID":"SHPIN40705",
        "Shipdate" : "2021-06-28",
        "Order comment" :"good old customer back"
        }


    },
    {
        "ID": "GABCA07324",
        "Name": "Salma          Begum",
        "Gender": "F",
        "Address": "Road 21, Gulshan",
        "City": "Dhaka",
        "Postcode": "2114",
        "Phone":"8801592457044",
        "Order":
        {
        "Order ID" :"ORDRE49100",
        "Product ID" :"PRDTA00103",
        "Product name" :"Office chair",
        "Ordered product stock" : "10",
        "Order product cost" : "40000",
        "Employee ID" : "EMPLE24598",
```

```
                "Employee name" :"Bokul          Ahmed",
                "Order date" : "2021-07-07",
                "Shipment_ID":"SHPIN55555",
                "Shipdate" : "2021-07-09",
                "Order comment" :"customer wants to track this"
                }

        },
        {
                "ID": "HABCA00024",
                "Name": "Sultan          Ahmed",
                "Gender": "M",
                "Address": "Road 2, Gulshana",
                "City": "Dhaka",
                "Postcode": "2141",
                "Phone":"8801792458030",
                "Order":
                [
                {
                "Order ID" :"ORDRE45679",
                "Product ID" :"PHGTA00144",
                "Product name" :"Office table",
                "Ordered product stock" : "1",
                "Order product cost" : "6000",
                "Employee ID" : "EMPLE24598",
                "Employee name" :"Bokul          Ahmed",
                "Order date" : "2021-05-24",
                "Shipment_ID":"SHPIN30078",
                "Shipdate" : "2021-05-26",
                "Order comment" :"customer felt weird"
                },
                {
                "Order ID" :"ORDRE45680",
                "Product ID" :"PRDTA00103",
                "Product name" :"Office chair",
                "Ordered product stock" : "1",
                "Order product cost" : "4000",
                "Employee ID" : "EMPLE24598",
                "Employee name" :"Bokul          Ahmed",
                "Order date" : "2021-05-24",
                "Shipment_ID":"SHPIN30079",
                "Shipdate" : "2021-05-26",
                "Order comment" :"customer felt weird"
                }
                ]
```

```
            }
            ]
            }
    ]
    )
```

Reference:

- Javatpoint, 2021, *Types of Databases*, Javatpoint, 6th November 2021, <https://www.javatpoint.com/types-of-databases>
- Lithmee, 2019, *What is the Difference Between Database and Data Structure*, Pediaa, 6th November 2021, <https://pediaa.com/what-is-the-difference-between-database-and-data-structure/>
- MariaDB, 2020, *Storage Engine Index Types*, MariaDB, 6th November 2021, <https://mariadb.com/kb/en/storage-engine-index-types/>
- Jonathan A.,2016**,** *A Quick Primer on Isolation Levels and Dirty Reads* , Infoq, 6th November 2021,<https://www.infoq.com/articles/Isolation-Levels/>
- MongoDB,2021, *Read Isolation, Consistency, and Recency* , MongoDB, 6th November 2021,<https://docs.mongodb.com/manual/core/read-isolation-consistency-recency/>
- MongoDB,2021, *What is NoSQL?* , MongoDB, 6th November 2021,<https://www.mongodb.com/nosql-explained>
- Ado Kukic,2020, *Set Global Read and Write Concerns in MongoDB 4.4* , MongoDB, 6th November 2021,<https://www.mongodb.com/developer/how-to/global-read-write-concerns/>
- MongoDB,2021, *Write Concern* , MongoDB, 6th November 2021,<https://docs.mongodb.com/manual/reference/write-concern/>
- MongoDB,2021, *Perform Two Phase Commits* , MongoDB, 6th November 2021, <https://docs.huihoo.com/mongodb/3.4/tutorial/perform-two-phase-commits/index.html>
- MariaDB,2021, *MariaDB Database Security* , MariaDB, 6th November 2021, <https://mariadb.com/database-topics/security/>
- MariaDB,2021, *MaxScale Filters Masking* , MariaDB, 6th November 2021, <https://mariadb.com/kb/en/mariadb-maxscale-21-masking/>
- MongoDB,2021, *NoSQL Database Security* , MongoDB, 6th November 2021, <https://www.mongodb.com/scale/nosql-database-security>
- David, W. and Edgar O.,2021, *DDoS Attack Trends for 2020* , F5 Labs, 6th November 2021, <https://www.f5.com/labs/articles/threat-intelligence/ddos-attack-trends-for-2020>
- MariaDB,2021, *SHOW PROFILE* , MariaDB, 6th November 2021, <https://mariadb.com/kb/en/show-profile/>
- MariaDB,2021,MariaDB *Audit Plugin* , MariaDB, 6th November 2021, <https://mariadb.com/kb/en/mariadb-audit-plugin/>
- MariaDB,2021, *Authentication from MariaDB 10.4* , MariaDB, 6th November 2021, <https://mariadb.com/kb/en/authentication-from-mariadb-104/>
- Jon C.,2016, *The Key Responsibilities of a Database Administrator* , DSP explorer, 6th November 2021, <https://content.dsp.co.uk/the-key-responsibilities-of-a-database-administrator>

Custom Database Report:

Name: SM Ragib Rezwan

ID: 103172423

Content List:

Overview of the database:

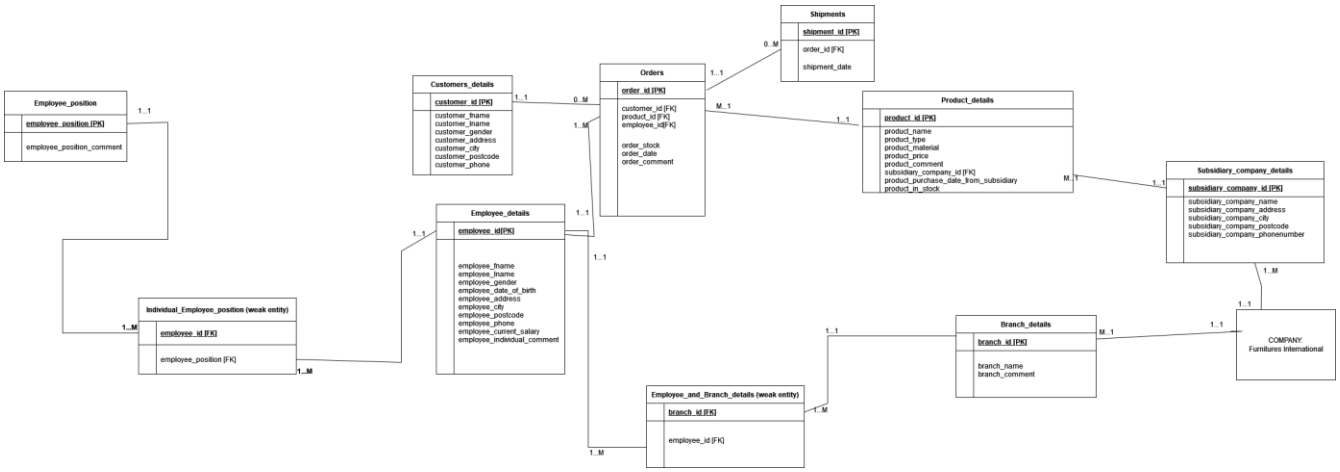
Background of company owning the database:

This is a database made for the Company called Furnitures International. Although it may seem like any other ordinary furniture selling shop, its current situation is rather serious. Previously they had been a big company, controlling a significant portion in the furniture market industry in Southeast Asia with branches in all relevant countries. But unfortunately, they had become a victim of a hostile takeover, losing almost all of their business and employees in the process. Furthermore, they were unable to cope with Covid-19 Crisis which led to further losses in personnel and assets. Now, they are currently trying to build up their business from scratch, trying to make the connections and gather employees using their last 2 branches and few subsidiary companies that they still have connection with.

Main use of the database:

This database will mainly be used to keep track of employee information, Customer information, product information, subsidiary company information, order details and shipping. Thus I used MYSQL here as it is highly structural, provides robust transactions following ACID(atomic, consistent, isolated, durable), etc. Furthermore, data can also be quickly updated and inserted into relevant tables via short queries by using MYSQL.

Simplified UML Diagram :



Tables, attributes, data types and the reasoning behind them:

a) Table Product

Attribute name	Datatype	Null or not	Justification
product_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all products must have an id so kept it not null
product_name	Varchar(20)	NOT NULL	The name is made of letter so kept it varchar to save space and also allow alphabet input. Also all products must have a name so kept it not null
product_type	Varchar(8)	NOT NULL	Fixed product type to only let chair, table, sofa, wardrobe as input. So kept it Varchar as letter input and also to save space. All products have a type so kept it as not null
product_material	Varchar(7)	NOT NULL	Fixed product type to only let wood, plastic, glass as input. So kept it Varchar as letter input and also to save space. All products are made of a material so kept it as not null
product_price	Int(6) UNISGNED	NOT NULL	Price of products sold by company is in int and wont go above 6 digits, so kept it as int(6). Also it cant be negative so unsigned. All products have a price, so kept it as not null
product_comment	Varchar(50)	NULL	This is the individual comment about the product so kept it varchar as letter and to save space. Product may or may not have a comment so kept it as null
subsidiary_company_id	Varchar(10)	NOT NULL	This is the id of the subsidiary company product was purchased from. So kept it as varchar for alphanumeric input and also to save space. All products have a subsidiary company id so kept it as not null
product_purchase_date_from_subsidiary	Date	NOT NULL	This is the date product was the last purchased from the subsidiary company. So kept it as date type. All products have a date when it had been last purchase from subsidiary company
Product_in_stock	Int (3)	NULL	Kept it int 3 as company currently don't have

			storage to store more than 999 product of a type. Also made it Nullable as some products may be newly introduced and thus they may not have stock for it yet. (alternatively, it can be kept as not null, but then 0 has to be kept as default for the new product introduction case)

Primary key name: product_id

Foreign key names: subsidiary_company_id

b) Table subsidiary_company_details

Attribute name	Datatype	Null or not	Justification
Subsidiary_company_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all company must have an id so kept it not null
Subsidiary_company_name	Varchar(20)	NOT NULL	The name is made of letter so kept it varchar to save space and also allow alphabet input. Also all company must have a name so kept it not null
Subsidiary_company_address	Varchar(40)	NOT NULL	Address is made of letter and numbers, so Varchar as alphanumeric input and also to save space. All companies have their own addresses and so not null
Subsidiary_company_city	Varchar(20)	NOT NULL	City is made of letter, so Varchar as letter input and also to save space. All companies have their own city and so not null
Subsidiary_company_postcode	Int(4) UNSIGNED	NOT NULL	PostCode is made of numbers, so int as number input and restricted it to 4 digits as postcode only have 4 digits. All companies have their own postcode and so not null
Subsidiary_company_phonenumber	Bigint(13) UNISGNED	NOT NULL	Phonenumber is made of numbers, so made it bigint as number input and 13 digits. Didn't use int as it can only take from 0 to 4294967295 when unsigned which is not enough! All companies have their own number and so not null

Primary key name: subsidiary_company_id

c) Table shipments

Attribute name	Datatype	Null or not	Justification
shipment_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all shipment must have an id so kept it not null
Order_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all shipment must have an order_id to correspond to it, so kept it not null
Shipment_date	Date	NOT NULL	Ship date is bsacially a date so kept it as date type. All shipments have their own shipping date to kept it not null

Primary key name: shipment_id

Foreign key names: order_id

d) Table Orders

Attribute name	Datatype	Null or not	Justification
order_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all orders must have an order id so kept it not null
customer_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all orders must have an customer id so kept it not null
product_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all orders must have a product id so kept it not null
employee_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all orders must have an employeer id so kept it not null
order_stock	Int(3)	NOT NULL	Order Stock is basically number of products purchased to kept it int to allow number input. Also it it wont be above 3 digit as originally stock stored is not more than 3 digits Also all orders must have order_stock so not null
order_date	date	NOT NULL	Order date is basically the date product has been ordered. So kept it date type. Also all orders must have an order date to kept it as not null
Order_comment	Varchar(50)	NULL	Comment is combination of number and letters to kept it as varhar to save space and also allow alphanumeric input. An order may or may not have any comment so it can be null

Primary key name: order_id

Foreign key names: customer_id, product_id, employee_id

Table type: It is a weak entity table

e) Table Customers details

Attribute name	Datatype	Null or not	Justification
customer_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all customers must have an customer id so kept it not null
customer_fname	Varchar(20)	NOT NULL	The fname is basically the firstname of the customer so kept it as varchar to save space and also allow letter input. Also all customers must have firstname so kept it not null
customer_lname	Varchar(20)	NOT NULL	The lname is basically the lastname of the customer so kept it as varchar to save space and also allow letter input. Also all customers must have lastname so kept it not null
customer_gender	Varchar(1)	NOT NULL	Gender basically has 3 possiblities: Male(M), Female (F), Unknown(U). So restricted its input to M,F,U Also all customers must have gender so kept it not null
customer_address	Varchar(20)	NOT NULL	The address is basically the address of the customer so kept it as varchar to save space and also allows letter input. Also all customers must have address so kept it not null
customer_city	Varchar(20)	NOT NULL	The city is basically the city of the customer so kept it as varchar to save space and also allows letter input. Also all customers must have city so kept it not null
customer_postcode	Int(4)	NOT NULL	The postis basically the postcode of the customer living area. So made it int and restricted input to 4 digits. Also all customers must have postcode so kept it not null
customer_phone	Bigint(13)	NOT NULL	Phonenumber is made of numbers, so made it bigint as number input and 13 digits. Didn't use int as it can only take from 0 to 4294967295 when unsigned which is not enough! All customers have their own number and so not null

Primary key name: customer_id

f) Table Employee details

Attribute name	Datatype	Null or not	Justification
employee_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all employee must have an customer id so kept it not null
employee _fname	Varchar(20)	NOT NULL	The fname is basically the firstname of the employee so kept it as varchar to save space and also allow letter input. Also all employee must have firstname so kept it not null

employee _lname	Varchar(20)	NOT NULL	The lname is basically the lastname of the employee so kept it as varchar to save space and also allow letter input. Also all employee must have lastname so kept it not null
employee _gender	Varchar(1)	NOT NULL	Gender basically has 3 possiblities: Male(M), Female (F), Unknown(U). So restricted its input to M,F,U Also all employee must have gender so kept it not null
Employee_date_of_birth	date	NOT NULL	The date of birth is basically the birthday of the employee so kept it as date type. Also all employee must have birthday so kept it not null
employee _address	Varchar(20)	NOT NULL	The address is basically the address of the employee so kept it as varchar to save space and also allows letter input. Also all employee must have address so kept it not null
employee _city	Varchar(20)	NOT NULL	The city is basically the city of the employee so kept it as varchar to save space and also allows letter input. Also all employee must have city so kept it not null
employee _postcode	Int(4)	NOT NULL	The postcode is basically the postcode of the employee's living area. So made it int and restricted input to 4 digits. Also all employee must have postcode so kept it not null
employee _phone	Bigint(13)	NOT NULL	Phonenumber is made of numbers, so made it bigint as number input and 13 digits. Didn't use int as it can only take from 0 to 4294967295 when unsigned which is not enough! All employee have their own number and so not null
employee_current_salary	Int(10)	NULL	Salary basically is the amount of money employee currently earns per year Employee may have newly joined the company and thus he may not have received his salary yet and thus kept it null (alternatively, it is also possible to keep it as not null, but then we will need to keep default as 0)
employee_individual_comment	Varchar(50)	NULL	This is the individual comment about the employee so kept it varchar as letter and to save space. employee may or may not have a comment so kept it as null

Primary key name: employee_id

g) Table Individual Employee position

Attribute name	Datatype	Null or not	Justification
employee_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all employee must have an customer id so kept it not null
employee _position	Varchar(10)	NOT NULL	Position basically has 3 possiblities: Manager, technician, normal.

	So restricted its input to these and used vachar for letter input and also to save space Also all employee must have position so kept it not null

Foreign key names: employee_id, employee_position

Table type: It is a weak entity table

h) Table Employee_position

Attribute name	Datatype	Null or not	Justification
employee _position	Varchar(10)	NOT NULL	Position basically has 3 possiblities: Manager, technician, normal. So restricted its input to these and used vachar for letter input and also to save space Also all employee must have position so kept it not null
Employee_position_comment	Varchar(50)	NOT NULL	This tell details of what the employee at this position does. So kept it as varchar as alphanumeric input and also to save space. Also made this specific comment not null as tells details about the employee's position's responsibility And thus must have relevant information in it regarding all possible employee positions.

Primary key names: employee_position

i) Table Employee_and_Branch_details

Attribute name	Datatype	Null or not	Justification
Branch_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all branches must have a branch id so kept it not null
Employee_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all employes must have a employee id so kept it not null

Foreign key names: branch_id, employee_id

Table type: It is a weak entity table

j) Table Branch_details

Attribute name	Datatype	Null or not	Justification
Branch_id	Varchar(10)	NOT NULL	The Id is a combination of letter and number to kept it varchar to save space and also allow alphanumeric input. Also all branches must have a branch id so kept it not null
Branch_name	Varchar(20)	NOT NULL	The name is basically name of branch so kept it varchar to save space and also allow letter input Also all branches must have a branch name so kept it not null
Branch_comment	Varchar(20)	NULL	This is the individual comment about the branch so kept it varchar as letter and to save space. branch may or may not have a comment so kept it as null

Primary key names: branch_id

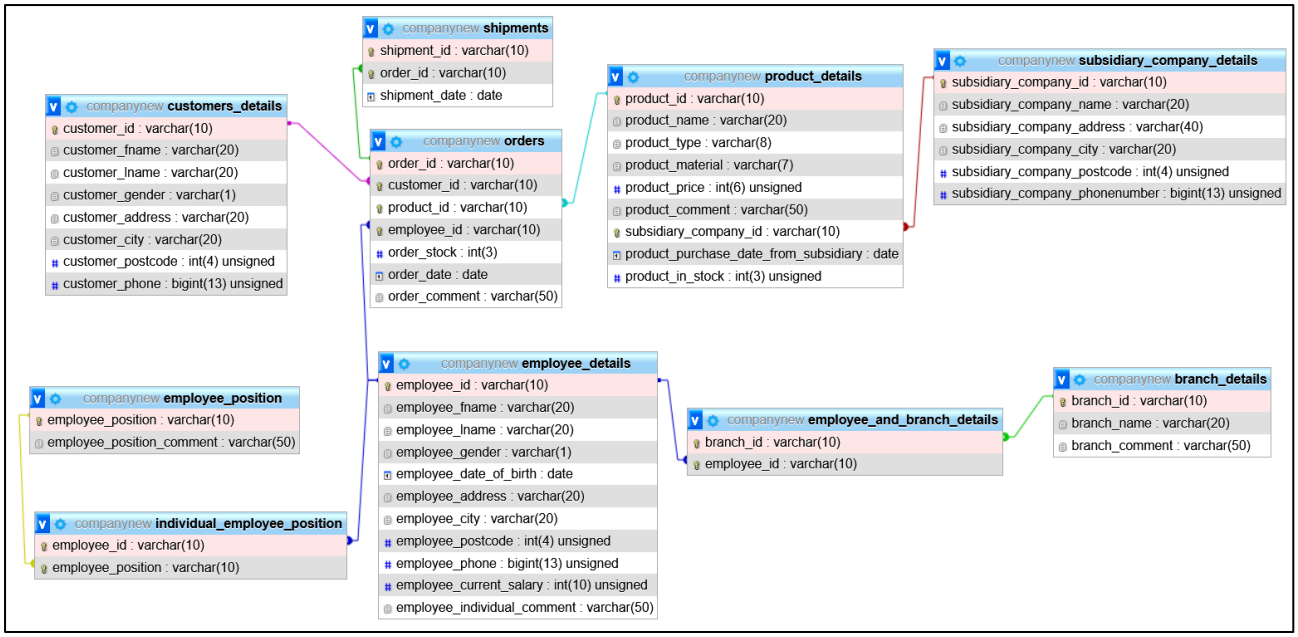
Scripts used to make the database:

```
CREATE database CompanyNew;
Use CompanyNew;

Create table Branch_details
(
branch_id varchar(10) NOT NULL,
branch_name varchar(20) NOT NULL,
branch_comment varchar(50),
 PRIMARY KEY(branch_id)
);

Create table Employee_details
(
employee_id varchar(10) NOT NULL ,
employee_fname varchar(20) NOT NULL,
employee_lname varchar(20) NOT NULL,
employee_gender varchar(1) NOT NULL
CHECK (employee_gender IN ('M','F', 'U')),
employee_date_of_birth date NOT NULL,
employee_address varchar(20) NOT NULL,
employee_city varchar(20) NOT NULL,
employee_postcode int(4) UNSIGNED NOT NULL,
employee_phone  BIGINT(13) UNSIGNED NOT NULL,
employee_current_salary int(10) UNSIGNED,
employee_individual_comment varchar(50),
```

```
PRIMARY KEY(employee_id)
);
Create table Employee_and_Branch_details
(
branch_id varchar(10) NOT NULL,
employee_id varchar(10) NOT NULL ,
PRIMARY KEY(branch_id , employee_id),
FOREIGN KEY(branch_id) REFERENCES Branch_details (branch_id),
FOREIGN KEY(employee_id) REFERENCES Employee_details (employee_id)
);
Create table Employee_position
(
employee_position varchar(10) NOT NULL
CHECK (employee_position IN ('Manager','Technician', 'Normal')),
employee_position_comment varchar(50) NOT NULL,
PRIMARY KEY(employee_position)
);
Create table Individual_Employee_position
(
employee_id varchar(10) NOT NULL ,
employee_position varchar(10) NOT NULL
CHECK (employee_position IN ('Manager','Technician', 'Normal')),
PRIMARY KEY(employee_id, employee_position),
FOREIGN KEY(employee_id) REFERENCES Employee_details (employee_id),
FOREIGN KEY(employee_position) REFERENCES Employee_position (employee_position)
);
Create table Subsidiary_company_details
(
subsidiary_company_id varchar(10) NOT NULL,
subsidiary_company_name varchar(20) NOT NULL,
subsidiary_company_address varchar(40) NOT NULL,
subsidiary_company_city varchar(20) NOT NULL,
subsidiary_company_postcode int(4) UNSIGNED NOT NULL,
subsidiary_company_phonenumber bigint(13) UNSIGNED NOT NULL,
PRIMARY KEY(subsidiary_company_id)
);
Create table Product_details
(
product_id varchar(10) NOT NULL ,
product_name varchar(20) NOT NULL,
product_type varchar(8) NOT NULL
CHECK (product_type IN ('chair', 'table', 'sofa', 'wardrobe')),
product_material varchar(7) NOT NULL
CHECK (product_material IN ('wood', 'plastic', 'glass')),
product_price int(6) UNSIGNED NOT NULL,
```

```
product_comment varchar(50),
subsidiary_company_id varchar(10) NOT NULL,
product_purchase_date_from_subsidiary Date NOT NULL,
product_in_stock int(3) UNSIGNED NOT NULL,
PRIMARY KEY(product_id ,subsidiary_company_id),
FOREIGN KEY(subsidiary_company_id) REFERENCES Subsidiary_company_details
(subsidiary_company_id)
);
Create table Customers_details
(
customer_id varchar(10) NOT NULL,
customer_fname varchar(20) NOT NULL,
customer_lname varchar(20) NOT NULL,
customer_gender varchar(1) NOT NULL
CHECK (customer_gender IN ('M','F', 'U')),
customer_address varchar(20) NOT NULL,
customer_city varchar(20) NOT NULL,
customer_postcode int(4) UNSIGNED NOT NULL,
customer_phone  BIGINT(13) UNSIGNED NOT NULL,
PRIMARY KEY(customer_id)
);
Create table Orders
(
order_id varchar(10) NOT NULL,
customer_id varchar (10) NOT NULL,
product_id varchar(10) NOT NULL,
employee_id varchar(10) NOT NULL ,
order_stock int(3) NOT NULL,
order_date date NOT NULL,
order_comment varchar(50),

PRIMARY KEY(order_id , customer_id ,product_id, employee_id),
FOREIGN KEY(customer_id) REFERENCES Customers_details (customer_id),
FOREIGN KEY(product_id) REFERENCES Product_details (product_id),
FOREIGN KEY( employee_id) REFERENCES Employee_details ( employee_id)
);
Create table Shipments
(
shipment_id varchar(10) NOT NULL,
order_id varchar(10) NOT NULL,
shipment_date date NOT NULL,
PRIMARY KEY(shipment_id  ,order_id),
FOREIGN KEY(order_id) REFERENCES Orders (order_id)
);
```

Name: SM RAGIB REZWAN ID: 103172423



Scripts used to insert data for all the tables:

```
INSERT INTO customers_details
 (customer_id, customer_fname, customer_lname, customer_gender,
  customer_address, customer_city, customer_postcode, customer_phone)
 VALUES ('HABCA00024', 'Sultan', 'Ahmed', 'M',
  'Road 2, Gulshan', 'Dhaka', 2141, 8801792458030);
  INSERT INTO customers_details
 (customer_id, customer_fname, customer_lname, customer_gender,
  customer_address, customer_city, customer_postcode, customer_phone)
 VALUES ('ADBCA00554', 'Kawsar', 'Hossain', 'M',
  'Road 10, Bonani', 'Dhaka', 4156, 8801796958971);
   INSERT INTO customers_details
 (customer_id, customer_fname, customer_lname, customer_gender,
  customer_address, customer_city, customer_postcode, customer_phone)
 VALUES ('FABCA09548', 'Robert', 'Brown', 'M',
  'Road 10, Modina', 'Sylhet', 3114, 8801892884308);
  INSERT INTO customers_details
 (customer_id, customer_fname, customer_lname, customer_gender,
  customer_address, customer_city, customer_postcode, customer_phone)
 VALUES ('ALBCA11020', 'Anne', 'Stuward', 'F',
  'Road 2, Rampura', 'Dhaka', 6915, 8801692433190);
  INSERT INTO customers_details
 (customer_id, customer_fname, customer_lname, customer_gender,
  customer_address, customer_city, customer_postcode, customer_phone)
 VALUES ('GABCA07324', 'Salma', 'Begum', 'F',
  'Road 21, Gulshan', 'Dhaka', 2114, 8801592457044);
INSERT INTO subsidiary_company_details
```

```
  (subsidiary_company_id, subsidiary_company_name, subsidiary_company_address,
subsidiary_company_city,
   subsidiary_company_postcode, subsidiary_company_phonenumber)
  VALUES ('SCOMP00001', 'Office Products', 'Road 49, Gulshan', 'Dhaka',
   2441, 8801972448031);
   INSERT INTO subsidiary_company_details
  (subsidiary_company_id, subsidiary_company_name, subsidiary_company_address,
subsidiary_company_city,
   subsidiary_company_postcode, subsidiary_company_phonenumber)
  VALUES ('SCOMP00444', 'Stylish Products', 'Road 2, Gulshan', 'Dhaka',
   2411, 8801973448031);
   INSERT INTO subsidiary_company_details
  (subsidiary_company_id, subsidiary_company_name, subsidiary_company_address,
subsidiary_company_city,
   subsidiary_company_postcode, subsidiary_company_phonenumber)
  VALUES ('SCOMP00051', 'RFL', 'Road 4,Bonani', 'Dhaka',
   6518, 8801745336711);
   INSERT INTO subsidiary_company_details
  (subsidiary_company_id, subsidiary_company_name, subsidiary_company_address,
subsidiary_company_city,
   subsidiary_company_postcode, subsidiary_company_phonenumber)
  VALUES ('SCOMP05001', 'Glassify', 'Road 4, Kumarpara', 'Sylhet',
   3141, 8801745336700);
     INSERT INTO product_details
  (product_id, product_name, product_type, product_material,
   product_price, product_comment, subsidiary_company_id,
product_purchase_date_from_subsidiary,product_in_stock)
  VALUES ('PRDTA00103', 'Office chair', 'chair', 'wood',
   4000, 'good office chair made of wood', 'SCOMP00001', '2020-12-12',70);
   INSERT INTO product_details
  (product_id, product_name, product_type, product_material,
   product_price, product_comment, subsidiary_company_id,
product_purchase_date_from_subsidiary, product_in_stock)
  VALUES ('PHGTA00144', 'Office table', 'table', 'wood',
   6000, 'good office table made of wood', 'SCOMP00001', '2019-2-16', 40);
   INSERT INTO product_details
  (product_id, product_name, product_type, product_material,
   product_price, product_comment, subsidiary_company_id,
product_purchase_date_from_subsidiary, product_in_stock)
  VALUES ('PRDTG00210', 'stylish chair', 'chair', 'wood',
   10000, 'good chair made of wood', 'SCOMP00444', '2020-12-15', 50);
   INSERT INTO product_details
  (product_id, product_name, product_type, product_material,
   product_price, product_comment, subsidiary_company_id,
product_purchase_date_from_subsidiary, product_in_stock)
```

```
 VALUES ('PRDTA00140', 'RFL sofa', 'sofa', 'plastic',
  500, 'good RFL chair made of plastic', 'SCOMP00051', '2018-10-12', 90);
  INSERT INTO product_details
 (product_id, product_name, product_type, product_material,
  product_price, product_comment, subsidiary_company_id,
product_purchase_date_from_subsidiary, product_in_stock)
 VALUES ('PROOA00103', 'clear wardrobe', 'wardrobe', 'glass',
  8000, 'good wardrobe made of glass', 'SCOMP05001', '2020-10-10', 90);
 INSERT INTO employee_details
 (employee_id, employee_fname, employee_lname, employee_gender,
employee_date_of_birth,
  employee_address, employee_city, employee_postcode, employee_phone,
employee_current_salary, employee_individual_comment)
 VALUES ('EMPLY44556', 'Karim', 'Kombol', 'M',
  '1990-10-24', 'Road 45, Gulshan', 'Dhaka', 2356, 8801731246924, 80000, 'good old
employee');
INSERT INTO employee_details
 (employee_id, employee_fname, employee_lname, employee_gender,
employee_date_of_birth,
  employee_address, employee_city, employee_postcode, employee_phone,
employee_current_salary, employee_individual_comment)
 VALUES ('EMPLE24598', 'Bokul', 'Ahmed', 'F',
  '1995-1-2', 'Road 40, Gulshan', 'Dhaka', 2557, 8801933246924, 60000, 'clumsy employee');
INSERT INTO employee_details
 (employee_id, employee_fname, employee_lname, employee_gender,
employee_date_of_birth,
  employee_address, employee_city, employee_postcode, employee_phone,
employee_current_salary, employee_individual_comment)
 VALUES ('ETYLE24594', 'Mokles', 'Miah', 'M',
  '2000-10-20', 'Road 45, Subidbazar', 'Sylhet', 3156, 8801731296924, 40000, 'Lazy
employee');
INSERT INTO employee_details
 (employee_id, employee_fname, employee_lname, employee_gender,
employee_date_of_birth,
  employee_address, employee_city, employee_postcode, employee_phone,
employee_current_salary, employee_individual_comment)
 VALUES ('QWRTY11111', 'Raima', 'Karim', 'F',
  '2001-3-4', 'Road 4, Modinamarket', 'Sylhet', 4156, 8801455672298, 80000, 'Diligent
employee');
INSERT INTO employee_details
 (employee_id, employee_fname, employee_lname, employee_gender,
employee_date_of_birth,
  employee_address, employee_city, employee_postcode, employee_phone,
employee_current_salary, employee_individual_comment)
 VALUES ('QSART45692', 'Kashem', 'Uddin', 'M',
```

```
  '1995-5-5', 'Road 8, Rampura', 'Dhaka', 2400, 8801459272298, 90000, 'old techy employee');
INSERT INTO branch_details
 (branch_id, branch_name, branch_comment)
 VALUES ('BRNCH40002', 'Dhaka Branch', 'Exclusive Branch in Dhaka');
 INSERT INTO branch_details
 (branch_id, branch_name, branch_comment)
 VALUES ('BRNCH20003', 'Sylhet Branch', 'Exclusive Branch in Sylhet');
INSERT INTO employee_and_branch_details
 (branch_id, employee_id)
 VALUES ('BRNCH40002', 'EMPLY44556');
  INSERT INTO employee_and_branch_details
 (branch_id, employee_id)
 VALUES ('BRNCH40002', 'EMPLE24598');
  INSERT INTO employee_and_branch_details
 (branch_id, employee_id)
 VALUES ('BRNCH20003', 'ETYLE24594');
 INSERT INTO employee_and_branch_details
 (branch_id, employee_id)
 VALUES ('BRNCH20003', 'QWRTY11111');
 INSERT INTO employee_and_branch_details
 (branch_id, employee_id)
 VALUES ('BRNCH40002', 'QSART45692');
INSERT INTO employee_position
 (employee_position, employee_position_comment)
 VALUES ('Manager', 'responsible for controlling group of staff');
 INSERT INTO employee_position
 (employee_position, employee_position_comment)
 VALUES ('Technician', 'maintains technical equipment');
  INSERT INTO employee_position
 (employee_position, employee_position_comment)
 VALUES ('Normal', 'ordinary person employeed');
 INSERT INTO individual_employee_position
 (employee_id, employee_position)
 VALUES ('EMPLY44556', 'Manager');
 INSERT INTO individual_employee_position
 (employee_id, employee_position)
 VALUES ('EMPLE24598', 'Normal');
 INSERT INTO individual_employee_position
 (employee_id, employee_position)
 VALUES ('ETYLE24594', 'Normal');
 INSERT INTO individual_employee_position
 (employee_id, employee_position)
 VALUES ('QWRTY11111', 'Manager');
 INSERT INTO individual_employee_position
 (employee_id, employee_position)
```

```
 VALUES ('QSART45692', 'Technician');
  INSERT INTO orders
 (order_id, customer_id, product_id, employee_id, order_stock, order_date, order_comment)
 VALUES ('ORDRE45679',          'HABCA00024',  'PHGTA00144',  'EMPLE24598',1,
         '2021-5-24',      'customer felt weird');
INSERT INTO orders
 (order_id, customer_id, product_id, employee_id, order_stock, order_date, order_comment)
 VALUES ('ORDRE45680',          'HABCA00024',  'PRDTA00103',  'EMPLE24598',1,
         '2021-5-24',      'customer felt weird');
INSERT INTO orders
 (order_id, customer_id, product_id, employee_id, order_stock, order_date, order_comment)
 VALUES ('ORDRE44990',          'FABCA09548',  'PROOA00103',  'ETYLE24594',10,
         '2021-6-26',      'good old customer back');
INSERT INTO orders
 (order_id, customer_id, product_id, employee_id, order_stock, order_date, order_comment)
 VALUES ('ORDRE49100',          'GABCA07324',  'PRDTA00103',  'EMPLE24598',10,
         '2021-7-7',       'customer wants to track this');

INSERT INTO shipments
 (shipment_id, order_id, shipment_date)
 VALUES ('SHPIN30078',          'ORDRE45679',  '2021-5-26');
 INSERT INTO shipments
 (shipment_id, order_id, shipment_date)
 VALUES ('SHPIN30079',          'ORDRE45680',  '2021-5-26');
  INSERT INTO shipments
 (shipment_id, order_id, shipment_date)
 VALUES ('SHPIN40705',          'ORDRE44990',  '2021-6-28');
INSERT INTO shipments
 (shipment_id, order_id, shipment_date)
 VALUES ('SHPIN55555',          'ORDRE49100',  '2021-7-9');
```

Use of 5 joins:

Some of the joins I used in the database and their purposes: (added reasoning for the places where I made views and sides notes for normal joins)

1) Use Join to make a view with full employee details (including position and their branch names)

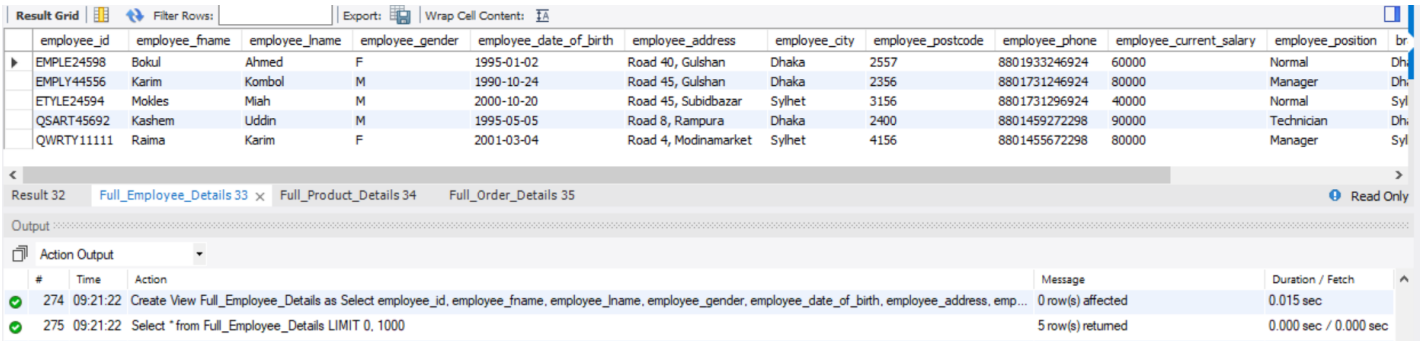
   Reason:
   To make it easier for employer to find all employee details in a single view instead of looking at several tables.

Also, since view is a stored query, it is already pre-optimized and thus would execute faster when opened next time as no need to re-optimize it (As query execution strategy has already been decided on by DBMS when the view was first created). Also, no need to use join and query through multiple tables again and again to find the data. Thus, overall it increases the performance of the database.

Commands:

```
Create View Full_Employee_Details as
Select employee_id, employee_fname, employee_lname, employee_gender,
employee_date_of_birth, employee_address,
employee_city, employee_postcode, employee_phone, employee_current_salary,
employee_position,branch_name, employee_individual_comment

 From employee_details e Natural Join employee_and_branch_details eb Natural Join
branch_details b NATURAL Join individual_employee_position ie NATURAL Join
employee_position
Order by employee_id;

Select *
from Full_Employee_Details;
```

Output:



employee_id	employee_fname	employee_lname	employee_gender	employee_date_of_birth	employee_address	employee_city	employee_postcode	employee_phone	employee_current_salary	employee_position	br
EMPLE24598	Bokul	Ahmed	F	1995-01-02	Road 40, Gulshan	Dhaka	2557	8801933246924	60000	Normal	Dh
EMPLY44556	Karim	Kombol	M	1990-10-24	Road 45, Gulshan	Dhaka	2356	8801731246924	80000	Manager	Dh
ETYLE24594	Mokles	Miah	M	2000-10-20	Road 45, Subidbazar	Sylhet	3156	8801731296924	40000	Normal	Syl
QSART45692	Kashem	Uddin	M	1995-05-05	Road 8, Rampura	Dhaka	2400	8801459272298	90000	Technician	Dh
QWRTY11111	Raima	Karim	F	2001-03-04	Road 4, Modinamarket	Sylhet	4156	8801455672298	80000	Manager	Syl

Result 32 | Full_Employee_Details 33 | Full_Product_Details 34 | Full_Order_Details 35 | Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
274	09:21:22	Create View Full_Employee_Details as Select employee_id, employee_fname, employee_lname, employee_gender, employee_date_of_birth, employee_address, emp...	0 row(s) affected	0.015 sec
275	09:21:22	Select * from Full_Employee_Details LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

2) Use Join to make a view for full product details (including subsidiary company name

Reason:

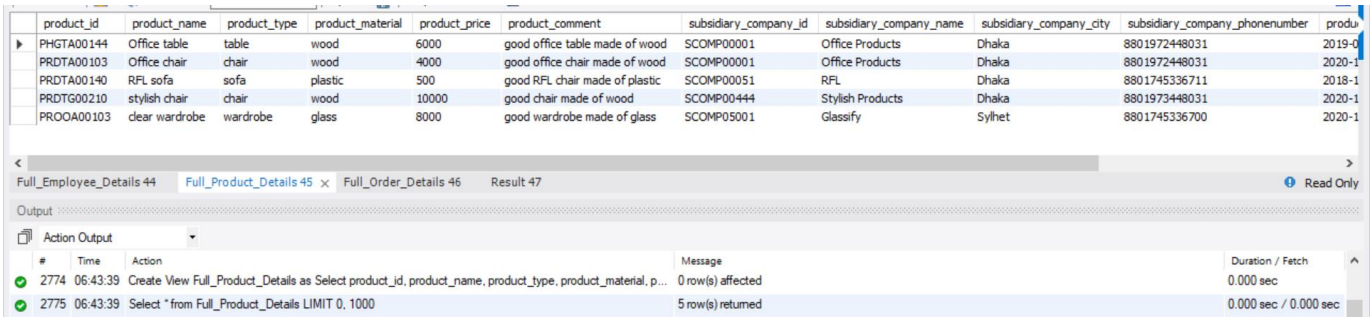
To have a single view with both poduct, subsidiary company information. This makes it easier for company to restock the products quickly, as they can easily see which company its from and the phone no of that company. Also there is company's city included and thus they can use it to estimate how long it will take for the product to be delivered to them.

Also, since view is a stored query, it is already pre-optimized and thus would execute faster when opened next time as no need to re-optimize it (As query execution strategy has already been decided on by DBMS when the view was first created). Also, no need to use join and query through multiple tables again and again to find the data. Thus, overall it increases the performance of the database.

Command:

```
Create View Full_Product_Details as
Select product_id, product_name, product_type, product_material, product_price,
product_comment, subsidiary_company_id,
subsidiary_company_name, subsidiary_company_city,
subsidiary_company_phonenumber, product_purchase_date_from_subsidiary,
product_in_stock
 From product_details pd Natural Join subsidiary_company_details scd
Order by product_id;

Select *
from Full_Product_Details;
```

Output:



product_id	product_name	product_type	product_material	product_price	product_comment	subsidiary_company_id	subsidiary_company_name	subsidiary_company_city	subsidiary_company_phonenumber	produ
PHGTA00144	Office table	table	wood	6000	good office table made of wood	SCOMP00001	Office Products	Dhaka	8801972448031	2019-0
PRDTA00103	Office chair	chair	wood	4000	good office chair made of wood	SCOMP00001	Office Products	Dhaka	8801972448031	2020-1
PRDTA00140	RFL sofa	sofa	plastic	500	good RFL chair made of plastic	SCOMP00051	RFL	Dhaka	8801745336711	2018-1
PRDTG00210	stylish chair	chair	wood	10000	good chair made of wood	SCOMP00444	Stylish Products	Dhaka	8801973448031	2020-1
PROOA00103	clear wardrobe	wardrobe	glass	8000	good wardrobe made of glass	SCOMP05001	Glassify	Sylhet	8801745336700	2020-1

Full_Employee_Details 44 | Full_Product_Details 45 | Full_Order_Details 46 | Result 47 | Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
2774	06:43:39	Create View Full_Product_Details as Select product_id, product_name, product_type, product_material, p...	0 row(s) affected	0.000 sec
2775	06:43:39	Select * from Full_Product_Details LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

3) Use Join to make a view for order stating name of employee and product and customer buying it

Reason:

To have a single view to see all order details including customer, employee who sold it, total cost of order, etc.

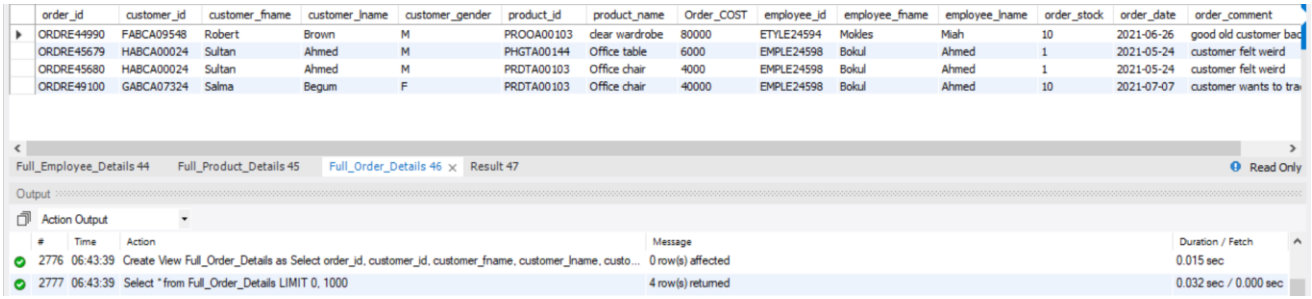
Also, since view is a stored query, it is already pre-optimized and thus would execute faster when opened next time as no need to re-optimize it (As query execution strategy has already been decided on by DBMS when the view was first created). Also, no need to use join and query through multiple tables again and again to find the data. Thus, overall it increases the performance of the database.

Command:

```
Create View Full_Order_Details as

Select order_id, customer_id, customer_fname, customer_lname, customer_gender,
product_id, product_name,  (SUM(product_price*order_stock)) as
Order_COST,employee_id,
employee_fname, employee_lname, order_stock, order_date, order_comment
 From orders o Natural Join product_details p  Natural Join customers_details c
Natural Join employee_details e
```

```
Group by order_id
Order by order_id;

Select *
from Full_Order_Details;
```

Output



order_id	customer_id	customer_fname	customer_lname	customer_gender	product_id	product_name	Order_COST	employee_id	employee_fname	employee_lname	order_stock	order_date	order_comment
ORDRE44990	FABCA09548	Robert	Brown	M	PROOA00103	clear wardrobe	80000	ETYLE24594	Mokles	Miah	10	2021-06-26	good old customer bac
ORDRE45679	HABCA00024	Sultan	Ahmed	M	PHGTA00144	Office table	6000	EMPLE24598	Bokul	Ahmed	1	2021-05-24	customer felt weird
ORDRE45680	HABCA00024	Sultan	Ahmed	M	PRDTA00103	Office chair	4000	EMPLE24598	Bokul	Ahmed	1	2021-05-24	customer felt weird
ORDRE49100	GABCA07324	Salma	Begum	F	PRDTA00103	Office chair	40000	EMPLE24598	Bokul	Ahmed	10	2021-07-07	customer wants to tra

Full_Employee_Details 44 Full_Product_Details 45 Full_Order_Details 46 Result 47 Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
2776	06:43:39	Create View Full_Order_Details as Select order_id, customer_id, customer_fname, customer_lname, custo...	0 row(s) affected	0.015 sec
2777	06:43:39	Select * from Full_Order_Details LIMIT 0, 1000	4 row(s) returned	0.032 sec / 0.000 sec

4) Use join to find which product was ordered more than once:

Commands:

```
Select order_id, product_id, product_name, (Count(product_id)) as Times_ordered
 From orders o Natural Join product_details p
group by product_id
Having Times_ordered>1;
```

Output:

order_id	product_id	product_name	Times_ordered
ORDRE45680	PRDTA00103	Office chair	2

Sidenote:

(We can also directly query from the “view” instead of using the joins, if that has been made beforehand. That will be more efficient and better as joins are expensive as it needs to look though many tables and compare datas before querying)

5) Use Join to find what the weird customer had ordered in details:

Command:

```
Select order_id, customer_id, customer_fname, customer_lname, product_id,
product_name,  (SUM(product_price*order_stock)) as Order_COST,
order_stock, order_date, order_comment
 From orders o Natural Join product_details p  Natural Join customers_details c
Group by order_id
Having customer_id = 'HABCA00024'
Order by order_id;
```

Output:

order_id	customer_id	customer_fname	customer_lname	product_id	product_name	Order_COST	order_stock	order_date	order_comment
ORDRE45679	HABCA00024	Sultan	Ahmed	PHGTA00144	Office table	6000	1	2021-05-24	customer felt weird
ORDRE45680	HABCA00024	Sultan	Ahmed	PRDTA00103	Office chair	4000	1	2021-05-24	customer felt weird

Sidenote:

(We can also directly query from the “view” instead of using the joins, if that has been made beforehand. That will be more efficient and better as joins are expensive as it needs to look though many tables and compare datas before querying)

Reference:

- Information from COS20015 (Fundamentals of database management) slide notes regarding MYSQL