Part A: Relational Database Design

For All the relations, we will define Primary key as the attributes which are remaining after removing all attributes that are in the right side of all FDs from the relation.

1) Author (Email, Name, Address, Telephone1, Telephone2, Telephone3)

Answer:
This relation is a result of an incorrect attempt to address multi-valued attribute problem.

It should be:
Author (Email, Name, Address, {Telephone})
The curly-braces denote the location is a multi-valued attribute.

So, this relation does not in 1NF. It is an unnormalised (or 0NF) relation. To address this problem, remove the offending attribute, along with Email and form a separate relation.

We'll then have:

Author (Email, Name, Address)
Author2 (Email*, Telephone)
(In Author2, multiple Telephones (for an author) are stored in multiple tuples.)
With no non-primary key attributes, we can safely assume that Author2 is in 3NF.

For Author,
FD1 : Email => Name, Address

For Author2,
FD1 : Email, Telephone => NIL

Email is the Primary Key. As all non-key attributes are fully functionally dependent on Primary key and no transitive functional dependency, Author is in 3NF.

2) Publisher (Name, Address, URL, ABN)

1. Identify functional dependencies among these attributes.

Answer:
FD1 : Name =>Address, URL, ABN

Name is the Primary key. As no transitive functional dependency and it meets the 2NF requirement, Publisher is in 3NF.

3) WrittenBy (Email*, ISBN*, Title*)

1. Identify functional dependencies among these attributes.

Answer:

FD1 : ISBN => Title

Primary key is <Email, ISBN>. As, Title is only dependent on ISBN, it is partially dependent on primary key, so it does not satisfy the 2NF condition and hence is in 1NF.

2. Decompose the above relation into a set of 3NF relations.

Answer:

We will split the relation into two.

WrittenBy (Email*, ISBN*)
WrittenBy1 (ISBN, Title*)

As, there is a relation "Book" already exists with ISBN primary key and Title, we do not need a relation "WrittenBy1".

For WrittenBy,
FD1 : Email, ISBN => NIL

With no non-primary key attributes, we can safely assume that WrittenBy is in 3NF.

4) Book (ISBN, Title, Edition, Year, ListPrice, PublisherName*)

1. Identify functional dependencies among these attributes.

Answer:

Book (ISBN, Title, Edition, Year, ListPrice, {PublisherName*})

Here, as per the rules, books can be published by one or more Publisher and hence PublisherName is a multi-value attribute, So this relation does not in 1NF. It is an unnormalised (or 0NF) relation. To address this problem, remove the offending attribute, along with ISBN and form a separate relation.

New relations,

Book (ISBN, Title, Edition, Year, ListPrice)
Book1 (ISBN*, PublisherName*)

For Book,
ISBN is the Primary key. As no transitive functional dependency and it meets the 2NF requirement, Book is in 3NF.

With no non-primary key attributes, we can safely assume that Book1is in 3NF.

5) Warehouse (Code, Address)

1. Identify functional dependencies among these attributes.

Answer:

FD1 : Code => Address

Code is the Primary key. As no transitive functional dependency and it meets the 2NF requirement, Warehouse is in 3NF.

6) StockedAt (ISBN*, Code*, StockQty)

1. Identify functional dependencies among these attributes.

Answer:

FD1 : ISBN, Code => StockQty

<ISBN, Code> is the Primary key. As non-key attribute is fully functionally dependent on Primary key and no transitive functional dependency requirement, StockedAt is in 3NF.

7) ShoppingCart (CartID, TimeStamp, ISBN, BuyPrice, Qty)

1. Identify functional dependencies among these attributes.

Answer:

FD1 : CartID => TimeStamp, ISBN, BuyPrice, Qty

CartID is the Primary key. As no transitive functional dependency and it meets the 2NF requirement, ShoppingCart is in 3NF.

8) Customer (Email, Name, Address, CartID*)

1. Identify functional dependencies among these attributes.

Answer:

FD1 : Email => Name, Address
FD2 : CartID => Email

CartID is the Primary key. Name & Address depends on Email. As there is a transitive dependency, we will split the relation into two,

Customer (Email, Name, Address)
Customer1 (CartID, Email*)

Here, it happened because Customer can have multiple Shopping carts and so foreign key should be on the many side. So, Instead of CartdID as a foreign key to Customer, there should be "Email" as a foreign key to ShoppingCart.

But, as we already have a relation with "CartID" as a primary key, we will combine Customer1 with ShoppingCart.

For Customer, As no transitive functional dependency and it meets the 2NF requirements, Customer is in 3NF.

Listing All Relational Database Schema as follows :-

Author (Email, Name, Address)
Author2 (Email*, Telephone)
Publisher (Name, Address, URL, ABN)
WrittenBy (Email*, ISBN*)
~~WrittenBy1 (ISBN, Title*)~~ // Redundant
Book (ISBN, Title, Edition, Year, ListPrice)
Book1 (ISBN*, PublisherName*)
Warehouse (Code, Address)
StockedAt (ISBN*, Code*, StockQty)
ShoppingCart (CartID, TimeStamp, ISBN, BuyPrice, Qty)
Customer (Email, Name, Address)
Customer1 (CartID, Email*)

As can be seen from above Relations, **WrittenBy1** is are redundant, as it is already covered by Book and **Customer1** & **ShoppingCart** has same primary key, so we will combine Customer1 with ShoppingCart.

After doing all the operations and renaming Author2 relation,

The Final Relational Database Schema looks like following :-

Author (Email, Name, Address)
Author-telephone (Email*, Telephone)
Publisher (Name, Address, URL, ABN)
WrittenBy (Email*, ISBN*)
Book (ISBN, Title, Edition, Year, ListPrice)
Book-publisher (ISBN*, PublisherName*)
Warehouse (Code, Address)
StockedAt (ISBN*, Code*, StockQty)
ShoppingCart (CartID, TimeStamp, ISBN, BuyPrice, Qty, Email*)
Customer (Email, Name, Address)

Part B: SQL

1) A) Using Sub Query :-

   SELECT firstname,lastname from author WHERE authorID In
   (SELECT distinct(authorID) from written_by where bookdescID IN (SELECT bookdescID from book where subjectID IN (SELECT subjectID from Subject WHERE subjecttype='DataBases')))

   B) Using JOINs:-

   SELECT aid.firstname,aid.lastname from author aid JOIN written_by wb ON aid.authorID=wb.authorID JOIN book b ON wb.bookdescID=b.bookdescID JOIN Subject s ON b.subjectID=s.subjectID WHERE s.subjecttype='DataBases'

2) SELECT aid.authorid,aid.firstname,aid.middlename,aid.lastname,b.title FROM author aid JOIN written_by wb ON aid.authorID=wb.authorID
   JOIN book b ON wb.bookdescID=b.bookdescID WHERE b.title LIKE 'American Electrician"s Handbook' and wb.role='Translator'

3) SELECT title from book where bookdescId NOT IN (SELECT distinct(bookdescID) from book_copy where bookID IN (SELECT bookID from borrow_copy))

4) A) SELECT title from book
   where bookdescId NOT IN (SELECT distinct(bookdescID) from book_copy where bookID IN (SELECT bookID from borrow_copy)) AND title LIKE '%DATABASE%'

B) SELECT title from book WHERE bookdescid IN (SELECT wb1.bookdescid from written_by wb join written_by wb1 on wb.authorid=wb1.authorid
where wb1.bookdescid <> wb.bookdescid AND wb.bookdescid=(SELECT bookdescid from book where title LIKE 'PRINCIPLES AND PRACTICE OF DATABASE SYSTEMS'))

5) SELECT pid.publisherfullname,b.title from publisher pid JOIN published_by pb ON pid.publisherID=pb.publisherID JOIN book b ON pb.bookdescID=b.bookdescID JOIN Subject s ON b.subjectID=s.subjectID WHERE s.subjecttype='DataBases'

6) A) select b1.title from book b1 left outer join borrow_copy bc1 on b1.bookdescid = bc1.bookid where bc1.transactionid is null

B) SELECT title from book where bookdescId NOT IN (SELECT distinct(bookdescID) from book_copy where bookID IN (SELECT bookID from borrow_copy))

7) SELECT pid.publisherfullname from publisher pid WHERE EXISTS (Select publisherID from published_by pb JOIN written_by wb ON pb.bookdescID=wb.bookdescID JOIN author aid ON wb.authorid=aid.authorid WHERE aid.firstname LIKE 'Alfred' AND aid.lastname LIKE 'Aho'

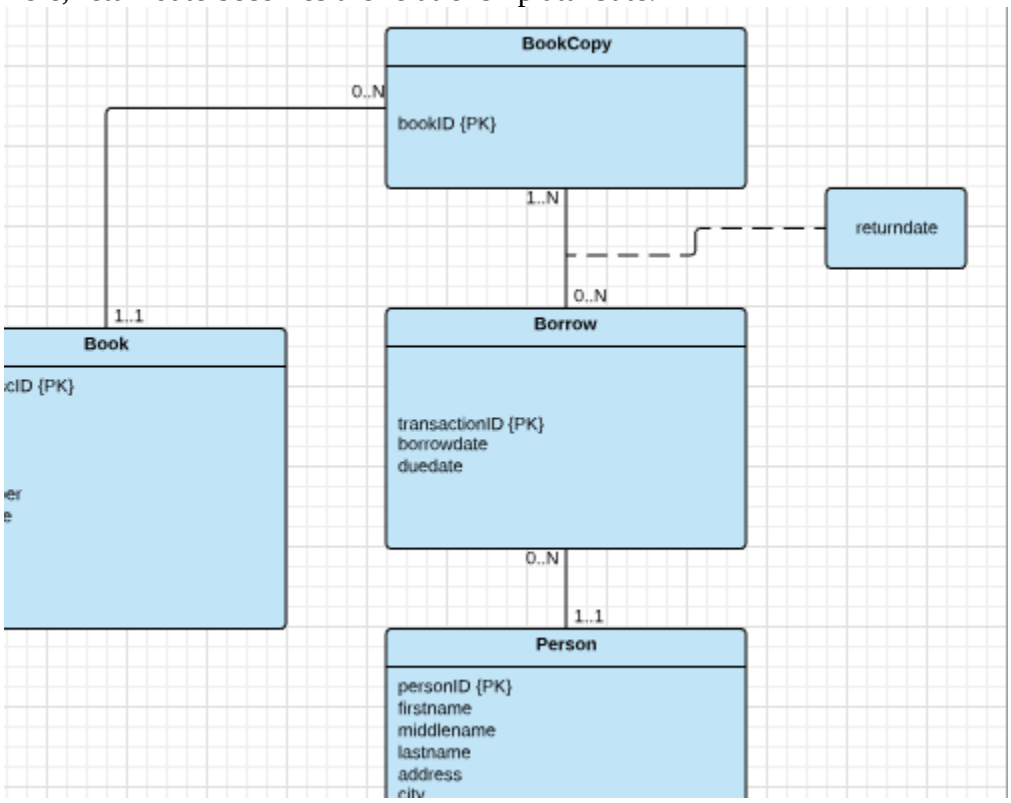
8) SELECT aid.firstname,aid.lastname,count(wb.bookdescid) FROM author aid JOIN written_by wb ON aid.authorid=wb.authorid GROUP BY aid.authorid HAVING count(wb.bookdescid)>3

9) SELECT b.title,count(b.bookdescid) from book b JOIN book_copy bc ON b.bookdescid=bc.bookdescid GROUP BY b.bookdescid HAVING count(b.bookdescid)=(SELECT max(count) FROM (SELECT b.title,count(bc.bookdescid) as count from book b JOIN book_copy bc ON b.bookdescid=bc.bookdescid GROUP BY b.bookdescid ORDER BY count(bc.bookid) DESC))

10) To return books borrowed in same transaction separately, Return Date should be in “Borrow_Copy” Table. So the records for each book of each transaction can be tracked with return date. Updated relational schemas are as follows :-

Borrow (transactionID, personID*, borrowdate, duedate)
borrow_copy (transactionID*, bookID*, returndate)

Here, returndate becomes the relationship attribute.



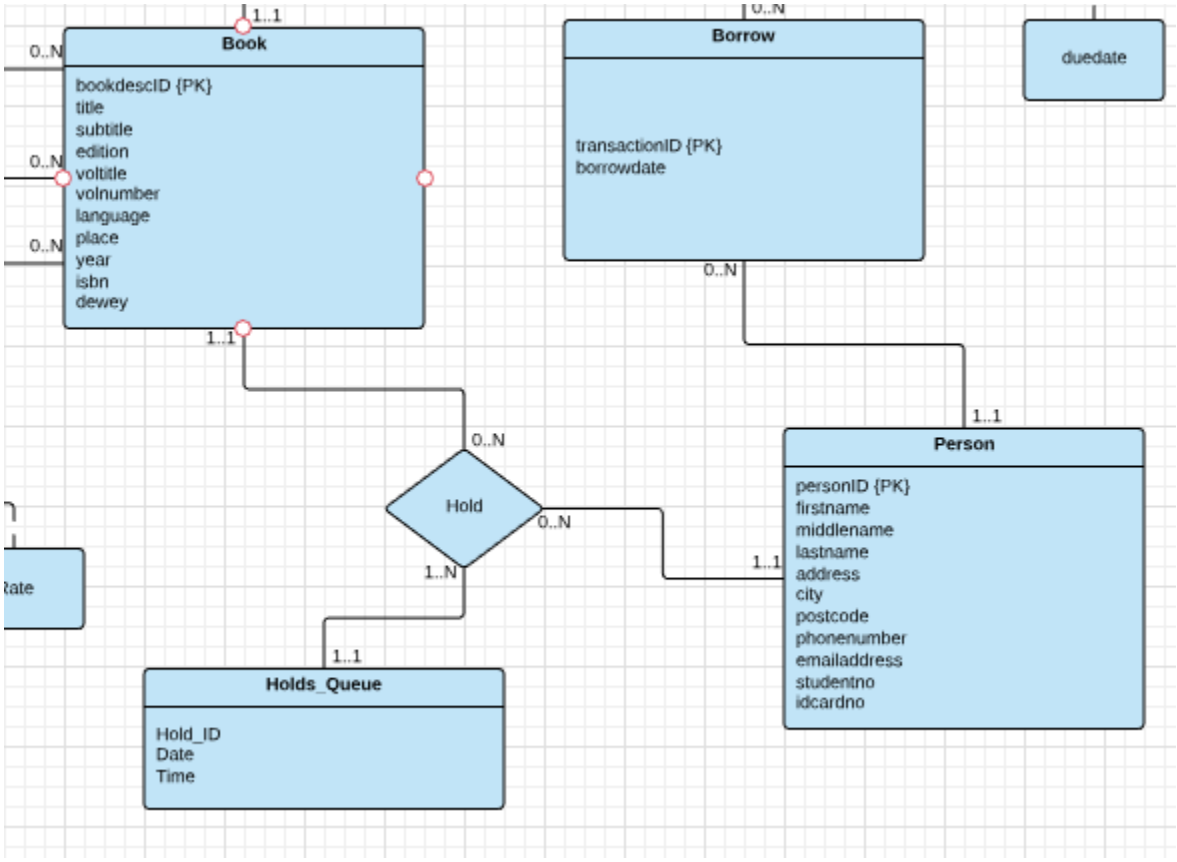
11) To add hold onto books, we can add a new relation to store the hold data. New relation schema is as follows :-

Holds_Queue (Hold_ID, Date, Time)

Then, We establish a ternary relationship among Book, Person and Holds_Queue, which creates A new relation,

Hold (Hold_ID*, bookdescId*, PersonId*)

Here, bookdescId, PersonId and Hold_ID combined makes a primary key. Date & Time determines the priority of the person who can borrow the book, which become available.

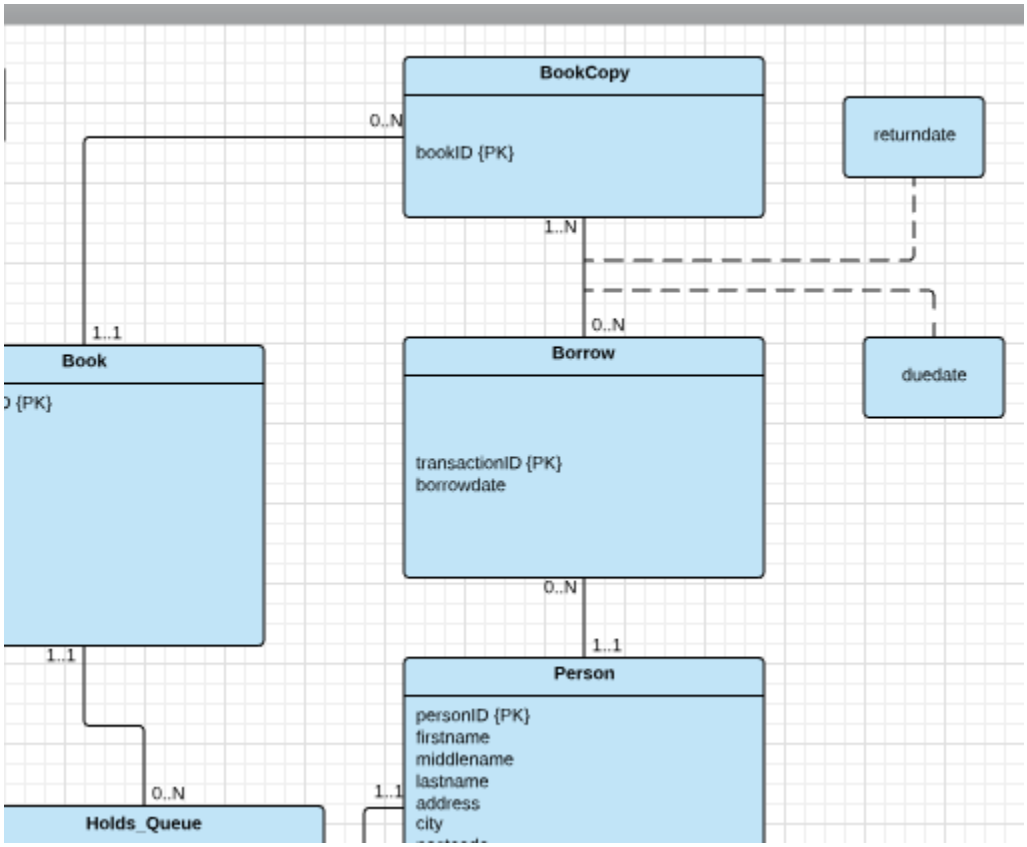


12) A) No, this database schema can't handle loan extensions for each book of the same transaction. (It can only handle Loan extension for Whole transaction, not individually for each book of a transaction)

To handle loan extension, we can shift "duedate" from Borrow to borrow_copy table and then it can be updated with the new due date for each book of each transaction.

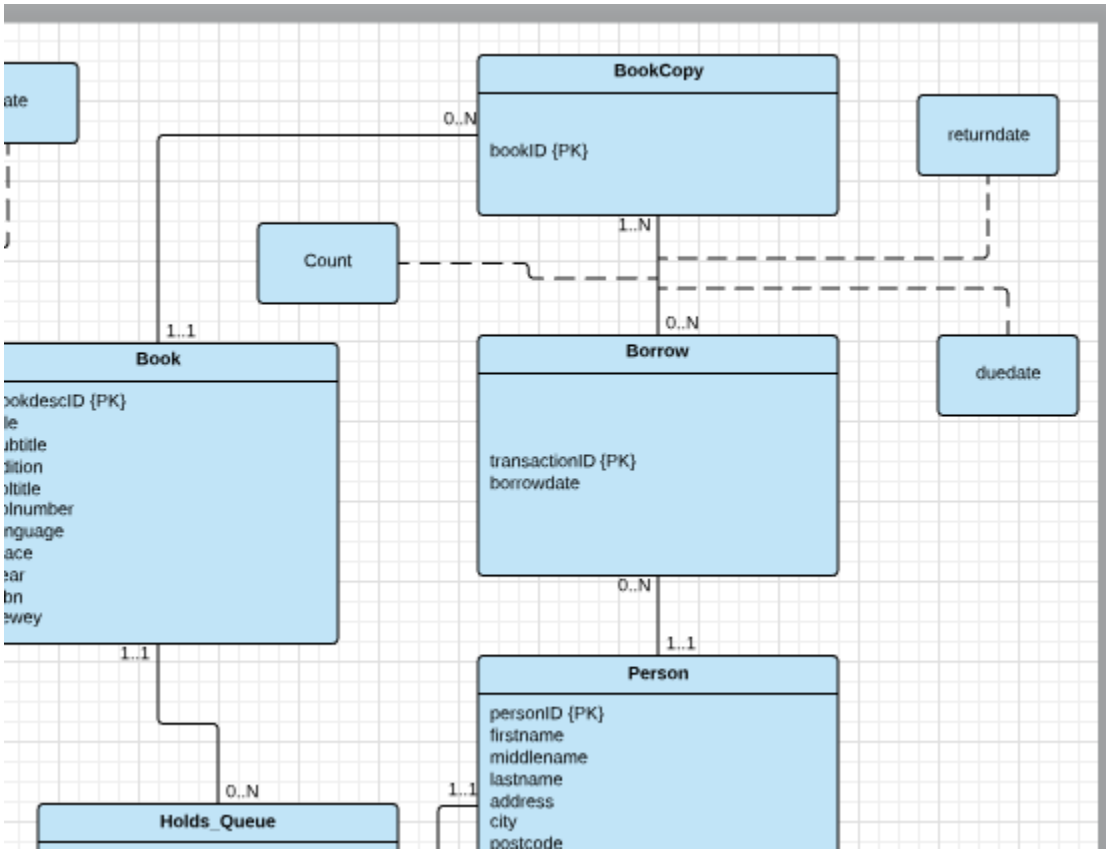
Updated relational schemas are as follows :-

borrow_copy (<u>transactionID*, bookID*</u>, duedate)



B) No, this database schema can't handle loan extensions only up to two times. To meet this requirement, we can shift "duedate" from Borrow to borrow_copy table and add a column named "Count" on Borrow table and set it to 0 when a record is created. Then create a trigger to check the value of Count column every time a due date value is updated, if it is 2 then it should not allow extend the due date, if not, increase the counter. Updated relational schemas are as follows :-

borrow_copy (<u>transactionID*, bookID*</u>, duedate, Count)



Part C: Research Question

I. Abstract
II. Motivation
III. Introduction
IV. Research & Analysis
V. Conclusion
VI. References

➢ Abstract:-

Ever since 1970s, Relational database has been the foundation of enterprise applications and it has been popular and inexpensive since the release of MySQL in 1995. Yet in recent years, an increased number of companies have adopted different types of non-relational database (MongoDB, Cassandra, Hypertable, Hbase/Hadoop, CouchDB etc), commonly referred to as NoSQL database. A NoSQL technology like MongoDB is not only used for new applications but also to augment or replace existing relational databases. This Research mainly focuses on one of the new technology of NoSQL database i.e. MongoDB, and makes a comparison study with one of the relational database i.e. MySQL and thus justifies why MongoDB is liked over MySQL. I will also describe the advantages and disadvantages of using relational & non-relational databases. A comparison criterion includes theoretical differences, characteristics, limitation, integrity, distribution, system requirements, and architecture, query and insertion times.

➢ Motivation:-

With the explosion in volume and variety of data due to increased mobile and web applications, the popularity of using an efficient database system to cater the needs of business has become an essential part. With that in mind, “Public Transport Victoria (PTV)” has made a decision to rebuild their Public transport timetable which is used to cater their website – “ptv.vic.gov.au” as well as mobile applications.

Public Transport Victoria is a Government website of Victoria State, Australia, which provides their users with the ability to access the daily time table of Buses, Trains and Trams throughout the State. With millions of users using this facility, PTV processes one million time tabling queries per day and forecasted a steady growth of 10% per year for next 5 years.

This Research is done to determine the best suitable technology, which caters the need to handle this large amount of data more efficiently, between Relational databases (MySQL) and NoSQL (MongoDB) databases.

➢ Introduction:-

Databases are defined as “collections of data”. Although when using the term database we refer to the complete database system, it refers only to the collection and data. “The system which handles Big data, transactions, problems, database engines, database schemas is called the Database Management System (DBMS)” [1].

In order to satisfy the need of storing and retrieving data, Databases were created. Different types of databases have invented their inception in the 1960’s, each using its own data representation and different technology for handling queries & transactions. “They began with navigational databases which were based on linked-lists, moved on to relational databases with joins, afterwards object-oriented and without joins in the late 2000s NoSQL(MongoDB, Cassandra, Hypertable, Hbase/Hadoop , CouchDB etc) emerged and has become a popular trend” [2].

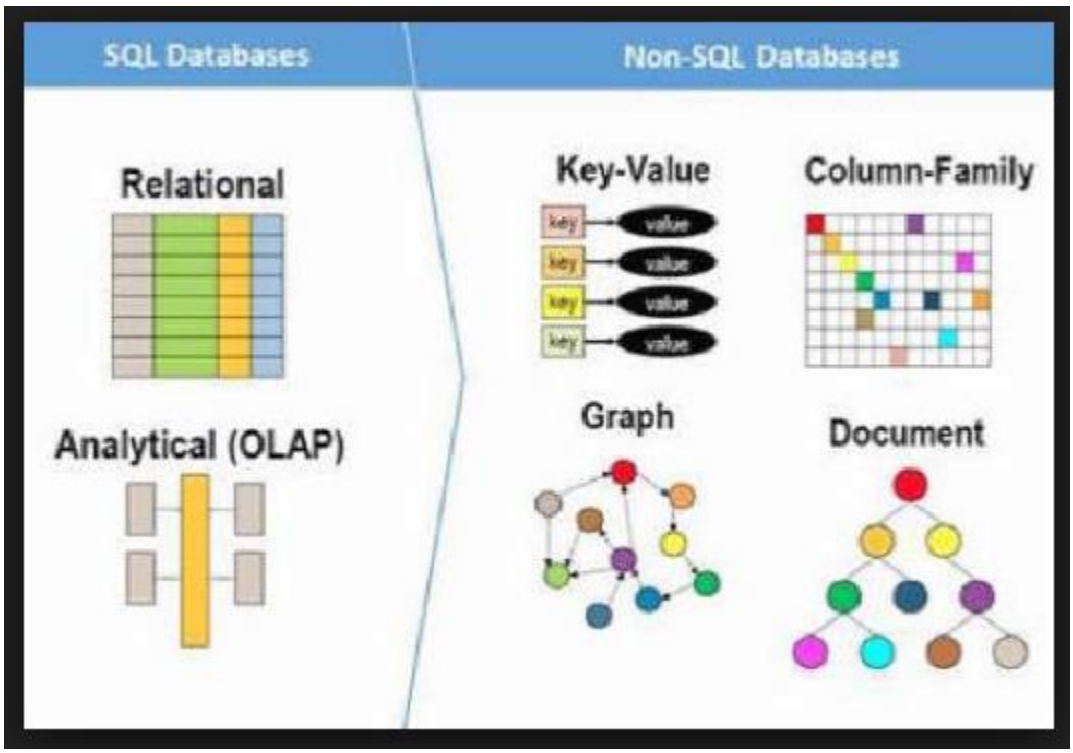


Fig-1 – Data Storing Strategies Comparison [7]

Relational databases are widely used in most of the applications and they exhibit great performance when they deal with a limited amount of data. For data with large volume like internet, multimedia and social media, traditional relational databases is ineffective. To overcome this problem the “NO SQL” term was introduced meaning, namely "Not Only SQL", which is a lenient variant of the term, compared to its previous significance, the anti-relational.

NoSQL is a methodology and not a tool, composed of many interdependent tools. The primary benefit of a NoSQL database is that, unlike a relational database, it can handle unstructured data such as documents, email, multimedia and social media efficiently. Non relational databases do not use the RDBMS principles (Relational Database Management System). As in Figure-1, there are four strategies for storing data in a non-relational database as follows:

1. Key-Value
2. Document : e.g. MongoDB

3. Column/Field
4. Graph-Oriented

Also, non-relational databases provide high flexibility for insertion or deletion of an attribute from the database because of the fact that they don't have a fixed database schema. In this research we concentrate on one of the NoSQL technologies, namely MongoDB, and make a comparison with MySQL to highlight why MongoDB is more capable than MySQL to cater the needs of "**Public Transport Victoria (PTV)**"

➢ Research & Analysis:-

- Below are three Figures, which compare the Terminology (Fig.-1), Features (Fig.-2) and Query Language (Fig.-3) used in both Relational (MySQL) and NoSQL (MongoDB) Databases:

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	`$lookup`, embedded documents
primary key	primary key
Specify any unique column or column combination as primary key.	In MongoDB, the primary key is automatically set to the _id field.
aggregation (e.g. group by)	aggregation pipeline See the SQL to Aggregation Mapping Chart.
transactions	transactions

Fig-1 – Terminology Comparison [3]

	MySQL	MongoDB	NoSQL Data Store
Open source	Yes	Yes	Yes
ACID Transactions	Yes	Yes	No
Flexible, rich data model	No	Yes	Partial: schema flexibility but support for only simple data structures
Schema governance	Yes	Yes	No
Expressive joins, faceted search, graphs queries, powerful aggregations	Yes	Yes	No
Idiomatic, native language drivers	No	Yes	No
Horizontal scale-out with data locality controls	No	Yes	Partial: no controls over data locality
Analytics and BI ready	Yes	Yes	No
Enterprise grade security and mature management tools	Yes	Yes	No
Database as a service on all major clouds	Yes	Yes	No

Fig-2 – Feature Comparison [4]

Fig-3 – Query Language Comparison [4]

Differences, Advantages, Disadvantages :

There are many differences between relational databases and NoSQL, all of them are important to understand before making a decision about best data management system. These include differences among:

- **Language**: SQL databases use SQL (Structured Query Language) for defining and manipulating data. This allows SQL to be extremely versatile and widely-used; however, it also makes it more restrictive. It requires that you use pre-defined schemas to determine the structure of your data before you even begin to work with it.

  A NoSQL database has a dynamic schema for unstructured data and the data can be stored in many different ways as stated in Introduction. This flexibility allows you to create documents without having to carefully plan and define the data structure or schema and add fields as you go.

- **Scalability:** Most SQL databases are vertically scalable, which means that you can increase the load on a single server by increasing components like CPU, SSD or RAM. On the other hand,

NoSQL databases are horizontally scalable, which means that they can handle more traffic/load simply by adding more servers to the database. They have the ability to become larger and much more powerful, which makes them the preferred choice for constantly evolving or large data sets. This caters the need of PTV to handle queries efficiently and powerfully with the expected annual increase over next five years without failing.

- **Community:** Due to SQL being in market for more than 40 years, it has a much larger, stronger and more developed community compared to NoSQL. There are thousands of chats and forums available where experts can share knowledge and discuss SQL best practices, which continuously enhance the skills. Although NoSQL is growing rapidly, its community is not as good as SQL due to the fact that it is still relatively new.

- **Structure:** SQL database are table-based which puts them as a better option for applications that require multi-row transactions. For Example, accounting systems or even legacy systems that were originally built for a relational structure. As stated earlier, NoSQL databases can be key-value pairs, wide-column stores, graph databases, or document-based.

NoSQL

Advantages

- **Non-Relational means table-less**
- **Mostly Open Source and Low-Cost**: An appealing solution for smaller organizations with limited budgets.
- **Easier scalability through support for Map Reduce**: NoSQL databases are designed to function on full throttle even with low-cost hardware.
- **No need to develop a detailed database model**: This saves a lot of development time.
- **Speed:** It's high-performing for simple queries, as all the related data are in single document which eliminates the join operations.

Disadvantages

- **Lack of reporting tools for analysis and performance testing**.
- **Lack of standardization in Query Language.**

Properties of Both: SQL databases follow ACID properties (Atomicity, Consistency, Isolation and Durability) while the NoSQL database emphasizes on the Brewers CAP theorem (Consistency, Availability and Partition tolerance).

- **Performance Comparison:** Below are the three figures which compare the performance of SQL and MongoDB using GUI Tool HeidiSQL 8.3 for MySQL, Robomongo 0.8.3 for MongoDB from a similar research paper. The cost time of MongoDB and MySQL for query speed were recorded as shown in graph by Authors.

Number of Parallel Clients 5 Time in seconds

	Total Rows	Rows / client	SQL Time	Mongo Time	Sql Ops/sec	Mongo Ops/sec
Basic Query	50	10	0.1	0.08	500	625
with index	500	100	0.38	0.1	1,316	5,000
	5,000	1,000	2.8	1.2	1,786	4,167
	25,000	5,000	14	4	1,786	6,250
	50,000	10,000	28	10.4	1,786	4,808

Fig-4 – Query Speed Comparison [6]

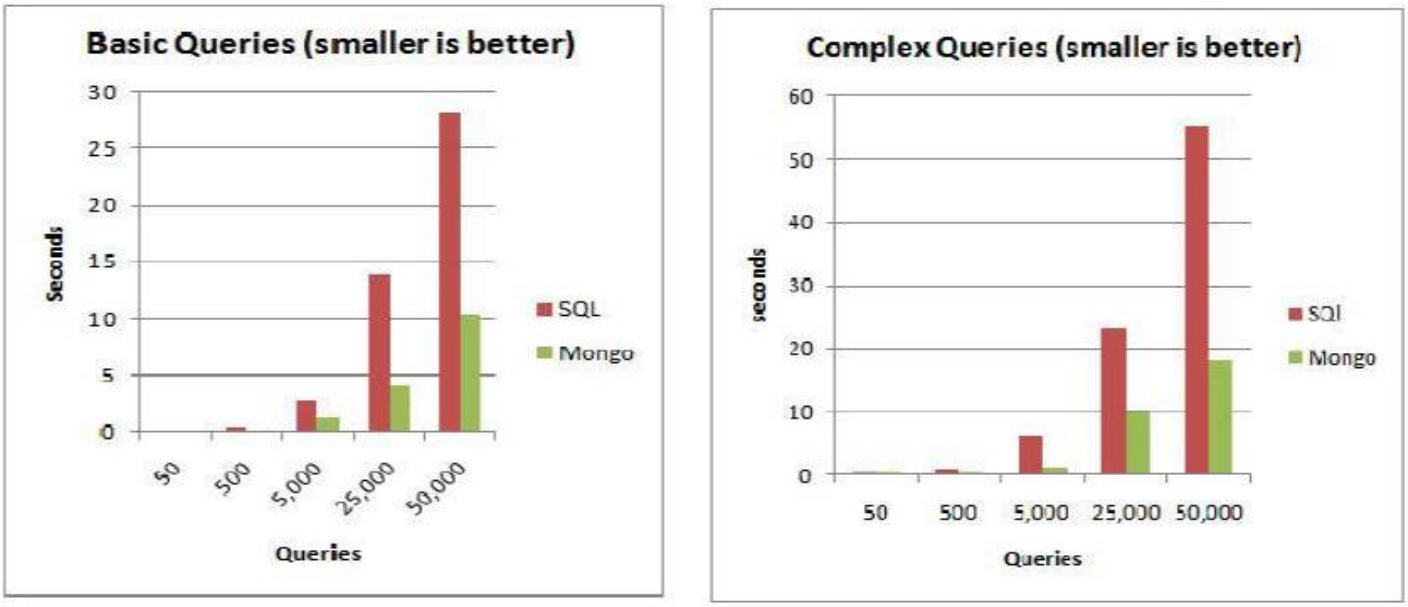


Fig-5 – Basic and Complex Query Time Comparison [6]

As can be seen from the graphs, performance of MongoDB is quite high compared to MySQL. Again, it depends on the operation performed and total records associated in operation. But in general, MongoDB has faster processing time.

To Sum Up:

- Relational Databases or MySQL is a good choice for any business who has pre-defined data structures and schemas. Relational databases are generally better for ACID level Transactions and for the systems whose schema doesn't change often.
- MongoDB or other NoSQL databases are used in the business with rapid growth or with the databases with no unambiguous schema definitions or if your application has large amount of queries to be performed in efficient way.

➢ Conclusions:-

"If your business is not going to experience significant growth in the near future and your data is structured, SQL is the right choice for your business. But if you desire rapid processing of data and you don't have transactional data to protect, NoSQL is your go-to solution" [5].

Keeping in mind the case study and performance evaluation for both paradigms, comparing above research & analysis and summary, as the **"Public Transport Victoria (PTV)"** don't deal with any Transactional level scenarios and it is expecting to have a significant growth in the future, and that needs a faster and efficient processing of large amount of data, I would recommend MongoDB as their backend database server.

➢ References:-

[1]. https://searchsqlserver.techtarget.com/definition/database-management-system

[2]. Berg K., Seymour T., and Coel R. History of Databases. International Journal of Management and Information Services, 17, 2013.

[3]. https://docs.mongodb.com/manual/reference/sql-comparison/

[4]. https://www.mongodb.com/compare/mongodb-mysql

[5]. https://blog.couchbase.com/comparison-sql-nosql-simplify-database-decision/

[6]. Lokesh Kumar1, Dr. Shalini Rajawat, Krati Joshi. "Comparative analysis of NoSQL (MongoDB) with MySQL Database", IJMTER – ISSN: 2349 – 9745

[7]. https://www.researchgate.net/figure/SQL-and-NoSQL-databases_fig3_299535734