ASSIGNMENT 2
DATABASE CONCEPTS

PART- A

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

In the above relation, we can see that Telephone is a multi-valued attribute. Hence it is in Unnormalized Form. It should be

Non Trivial Functional Dependencies:

FD1: Email -> Name

FD2: Email ->Address

Highest Normalized Form:

Author(Email, Name, Tele1, Tele2, Tele3)

Telephone is a multi valued so relation is not in 1NF

Decompose it into 2NF:

Author1(Email, Name, Address)

Author2(Email*, Telephone)

Now the both relations aren't any multi valued attributes,transitive dependencies so, they are in 3NF.

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

Non Trivial Functional Dependencies:

FD1: Name -> Address

FD1: Name ->URL

FD1: Name -> ABN

Highest Normalized Form:

It is in 3NF as there aren't any multi valued attributes, composite primary keys and transitive dependencies.

3) WrittenBy(Email, ISBN, Title)

Non Trivial Functional Dependencies

FD1: ISBN -> Title

Highest Normalized Form:

As Email cannot be determined by any of the remaining attributes in this relation, it should be considered as a part of primary key.
Hence the Primary key is **ISBN, Email**.
Written_By(ISBN, Email, Title)
As there are no multi-valued attributes the above relation is in 1NF as there. But, it is not in 2NF as Title is partially dependent on PK (As the Title is dependent on ISBN but not on Email)
Converting into 2 NF: We need to create separate relations as below to eliminate partial dependencies.

WrittenBy1(ISBN, Title)

WrittenBy2(ISBN*, Email)

the above relations are also in 3 NF.

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

Functional Dependencies:FD1: ISBN -> Title

FD1: ISBN -> Year

FD1: ISBN -> Edition

FD1: ISBN -> ListPrice

Here ISBN is primary key as it is unique

Highest Normalized Form:

Book(ISBN, Publishername Title, Year, Edition, Listprice)
It is in 1 NF, as there are partial dependencies.

Converting into 2NF:
Book_New1(ISBN, Title, Year, Edition, Listprice)
Book_New2(ISBN*, Publishername)
The above relations are in 2NF
As the above Relations does not have any transitive dependencies then they are in 3NF as well.

5) Warehouse(Code, Address)
Non Trivial Functional Dependencies:
Code -> Address
Highest Normalized Form:
Warehouse(Code, Address)
It is in 3NF as there aren't any multi valued attributes, composite primary keys and transitive dependencies.

6) StockedAt(ISBN, Code, StockQty)
Non Trivial Functional Dependencies:
FD1: ISBN, Code -> StockedQty
From the above functional dependency we can see that ISBN,CODE are primary keys.
Highest Normalized Form:
Stocked_At(ISBN, Code, StockQty)
It is in 3NF as there aren't any multi valued attributes, composite primary keys and transitive dependencies.
7) ShoppingCart(CartID, Timestamp, ISBN*, BuyPrice, Qty)
Non Trivial Functional Dependencies:
FD1: CartID->Timestamp
FD2: CartID, ISBN->Qty
FD3: CartID, ISBN->Buyprice
FD4: CartID->cust_email
From the above functional dependencies we see that primary keys are CartID,ISBN.
Highest Normalized Form:
Shopping_cart(CartID, ISBN, Timestamp, Qty, Buyprice, cust_email)
It is in 1 NF, as it has partial dependencies.

Converting into 2NF:

Shopping_cart1(CartID, Timestamp, cust_email)

Shopping_cart2(CartID*, ISBN, Qty, Buyprice)

There Are no transitive dependencies so its in 3NF.

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

Non Trivial Functional Dependencies:

FD1: cust_Email□cust_Name
FD2: cust_Email□Address
FD3: CartID □ cust_Email

From Functional dependencies we know that the Primary key is **CartID**.
Cust(CartID, cust_Email, cust_Name, Address)
As there are no multi-valued attributes and partial dependencies the relation is in 1NF and 2NF. But, it is not in 3NF as cust_Name and Address are transitively dependent on CartID.

Highest Normalized Form:

From Functional dependencies we know that the Primary key is **CartID**.
Cust(CartID, cust_Email, cust_Name, Address)
As there are no multi-valued attributes and partial dependencies the relation is in 1NF and 2NF. But, it is not in 3NF as cust_Name and Address are transitively dependent on CartID.
Decompose to 3NF:
Cust1(CartID, cust_Email*)
Cust2(cust_Email, Name, Address)
As there are no transitive dependencies the above relations are in 3NF.

Relations:

1..Author1(Email, Name,Address)

2. Author2(Email*, Telephone)

3. Publisher(Name, URL, Address, ABN)

4. Written_By1(ISBN, Title)

5. Written_By2(Email, ISBN*)

6. Book_New1(ISBN, Title, Year, Edition, Listprice)

7. Book_New2(ISBN*, Publishername)

8. Warehouse(Code, Address)

9. Stocked_At(ISBN, Code, StockQty)

10.Shopping_cart1(CartID,Timestamp,cust_email)

11.Shopping_cart2(CartID*, ISBN, Qty, Buyprice)

12.Cust1(CartID, cust_Email*)

13.Cust2(cust_Email, Name, Address)

Combining relations with common primary keys. The relation WrittenBy2 and Book2 have same primary keys. Hence combining them we get single relation. Renaming the relations into related names.

1.Author1(Email, Name,Address)

2.Author2(Email*, Telephone)

3.Publisher(Name, URL, Address, ABN)

4.Written_By2(Email,ISBN*)

5.Book_New1(ISBN, Title, Year, Edition, Listprice)

6.Book_New2(ISBN*, Publishername)

7.Warehouse(Code, Address)

8.Stocked_At(ISBN, Code, StockQty)

9.Shopping_cart1(CartID, Timestamp, cust_email)

10.Shopping_cart2(CartID*, ISBN, Qty, Buyprice)

11.Cust2(cust_Email, Name, Address)

PART-B

1. a)

```
SELECT a.firstname, a.lastname
FROM author a, book b, subject s, written_by W
WHERE s.subjecttype IN (SELECT s.subjecttype
                        FROM subject s
                        WHERE LOWER(s.subjecttype)='databases')
                         AND s.subjectID = b.subjectID
                         AND w.authorID = a.authorID
                         AND b.bookdescID = w.bookdescID;
```

b)

```
SELECT a.firstname, a.lastname
FROM author A JOIN written_by W ON a.authorID = w.authorID
JOIN book b ON w.bookdescID = b.bookdescID
JOIN subject s ON b.subjectID = s.subjectID
WHERE subjecttype = 'DataBases';
```

2.

```
SELECT a.firstname, a.middlename, a.lastname
FROM author a JOIN written_by w ON a.authorID = w.authorID
JOIN book b ON W.bookdescID = b.bookdescID
WHERE b.title = 'AMERICAN ELECTRICIAN''S HANDBOOK'
AND w.role = 'Translator';
```

3.

```
SELECT book.title as "Book Title"
FROM book b
WHERE b.bookdescID NOT IN (SELECT bc.bookdescID
                           FROM borrow_copy bwc
                LEFT OUTER JOIN book_copy bc ON bwc.bookid = bc.bookid);
```

4. a)

```
SELECT title
FROM book
WHERE title LIKE '%DATABASE%'
```

b)

```
SELECT title
FROM book
WHERE NOT EXISTS (SELECT * FROM book a, written_by wb, author a
WHERE book.bookdescID = wb.bookdescID
AND wb.authorID = a.authorID
AND book.TITLE != 'PRINCIPLES AND PRACTICE OF DATABASE
SYSTEMS')
```

5.

```
SELECT pu.publisherfullname, b.title
FROM publisher pu
JOIN published_by pb ON pu.publisherID = pu.publisherID
JOIN book b ON pb.bookdescID = b.bookdescID
JOIN subject sb ON b.subjectID = sb.subjectID
WHERE subjecttype = 'DataBases';
```

6. a)

```
SELECT book.title as "Book Title"
FROM book book
WHERE book.bookdescID NOT IN (SELECT bookcopy.bookdescID
                                    FROM borrow_copy bwc
                LEFT OUTER JOIN book_copy bookcopy ON bwc.bookid =
bc.bookid);
```

b)

```
SELECT b.title as ‘Book Title’;
FROM book b
WHERE b.bookdescID NOT IN ( SELECT b.bookdescID
                                        FROM (book b JOIN book_copy bc
                                        ON b.bookdescID = bc.bookdescID)
                                JOIN borrow_copy bwc ON bwc.bookID =
        bc.bookID)
```

7.

```
SELECT pu.publisherfullname
FROM publisher pu
JOIN published_by pb ON pu.publisherID = pb.publisherID
JOIN written_by wb ON pb.bookdescID = wb.bookdescID
JOIN author a ON wb.authorID = a.authorID
WHERE EXISTS (SELECT a.firstname, a.lastname
FROM author
WHERE a.firstname = 'ALFRED' AND a.lastname = 'AHO');
```

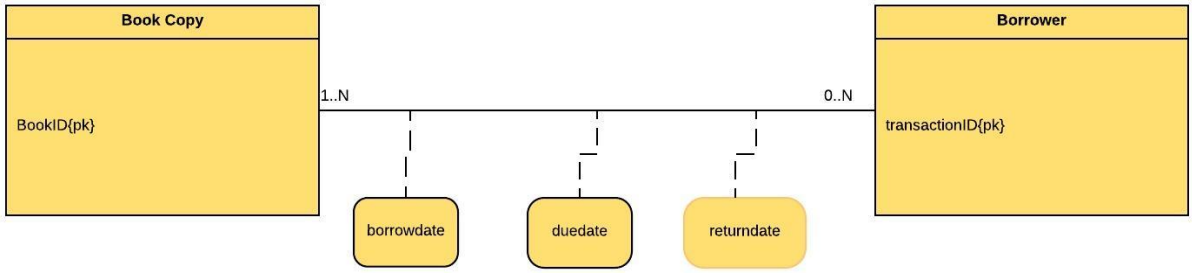
8.

```
SELECT a.firstname, a.lastname, COUNT(wb.authorID) AS 'Number of books'

FROM author a JOIN written_by wb ON a.authorID = wb.authorID

JOIN book b ON wb.bookdescID = b.bookdescID

GROUP BY a.authorID

HAVING COUNT(wb.authorID) > 3;
```

9.

```
SELECT book1.title, COUNT (bookcopy1.bookdescID) AS "No of book
copies"
FROM book book1 JOIN book_copy bookcopy1 ON book1.bookdescID=
bookcopy1.bookdescID
GROUP BY bookcopy1.bookdescID
HAVING COUNT(bookcopy1.bookdescID) = (SELECT MAX(y.number)
FROM ( select book.title, COUNT (bookcopy.bookdescID) AS "Number"
FROM book book JOIN book_copy bookcopy ON book.bookdescID =
bookcopy.bookdescID
GROUP BY bookcopy.bookdescID) AS y);
```

10.
Modified ER_Diagram



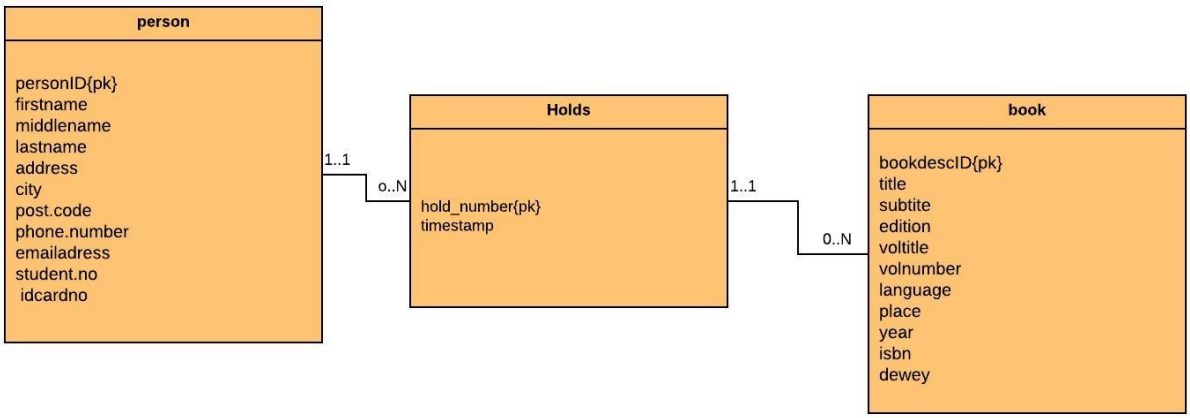
Modified Schema:

- BookCopy(bookID)
- Borrow(transactionID)
- Borrow_copy(bookID*,transactionID*,borrowdate,duedate,returnDate)

Explanation:

- According to modified schema and ER diagram “borrow_copy” includes the primary keys of both “BookCopy” and “Borrow” as foreign key, which together form composite primary key for borrow_copy relation.
- “borrow_copy” will include borrowdate,duedate, and also returndate attributes thereby recording borrowdate,duedate for each book return separately.

11 modified ER-diagram

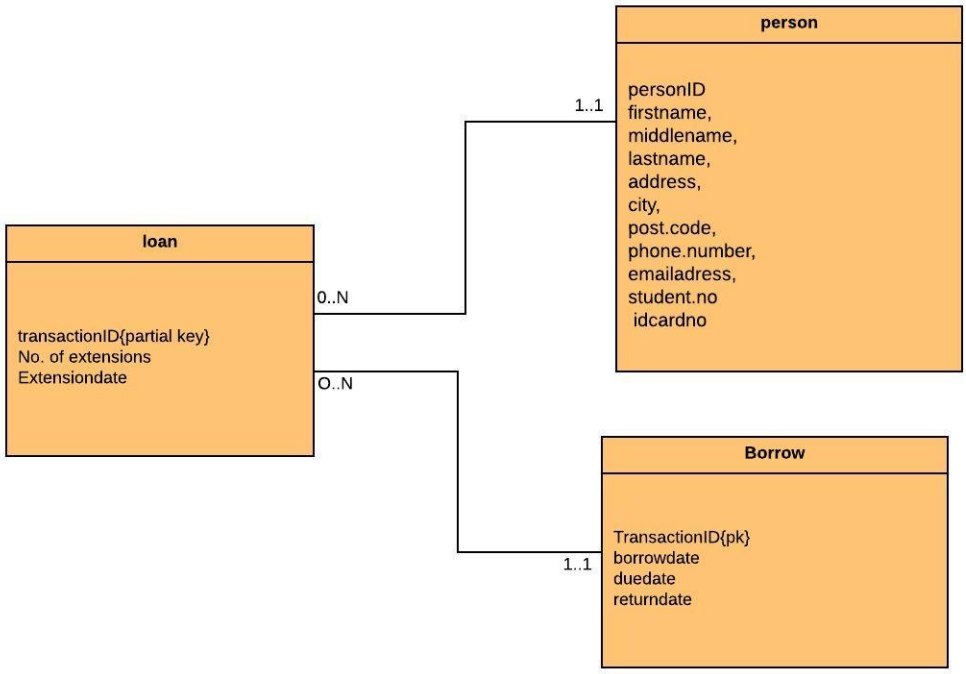


Modified Schema:

- Person(personID,firstname,middlename,lastname,address ,city,post.code,phone.number,emailadress,student.no, idcardno.)
- Book(bookdescID,title,subtite,edition,voltitle,volnumber, language,place,year,isbn,dewey)
- Holds(hold number,personID*,timestamp)

Explanation
For customer to place holds on books a new entity called holds is created which assigns a unique hold number tpo each hold placed and also records the time at which a hold has been placed.

12
Modified ER-diagram



Modified Schema:
Person(personID,firstname,middlename,lastname,address,city, post.code,phone.number,emailadress,student.no, idcardno.)
Borrow(transactionID,borrowdate,duedate,returndate)
Loan(PersonID*,TransactionID*, No. of extensions,Extensiondate)

Explaination:

- In order to accommodate loan extensions a new entity called loan is created which keeps the account of extensions done for a particular book by a person and also stores the date on which the extension is performed.Ω

PART-C

SQL AND NoSQL — A comparison between the relational database systems and NoSQL database systems.

1.Introduction:

Public Transport Victoria (PTV) has become a popular way to display train, tram and bus services. to multiple users across within Victoria. This research presents two methods for storing, retrieving, displaying and sharing data in the backend database server: 1) Traditional database system like SQL and Oracle 2) NoSQL database system like MongoDB. These two approaches are compared for cost, ease of creation and use, update speed, and shareability.

2. Situation Analysis:

In the given scenario, an assumption was made to rebuild the public transport timetable to meet the demand the popularity of current website and mobile users, it is also given from the description that on average one million timetabling queries are processed per day and is expected to have a surge of 10% annually. The task of the current report is finding the ideal approach which fits best into the given requirements.

3. SQL and NoSQL- a brief introduction:

SQL was Developed in the 1970s to manage data in applications, though NoSQL came into the spotlight in 2000s to manage restrictions of SQL databases, especially concerning scale, replication and unstructured information data storage.

For quite a long time Relational database (RDBMS), like SQL, has been commanding the business as the essential model for database administration. Anyway, with the ascent of social networks and million entry databases, non-relational "NoSQL" databases are picking up noticeable quality as an elective model for database administration.

SQL databases stay prominent as they fit normally into numerous admired programming stacks, including LAMP and Ruby-based stacks. These databases are surely known and upheld, which can be a noteworthy favourable position in

the event that you keep running into issues. Some of the most popular SQL databases are MySQL, Oracle, IMB DB2, Sybase, PostgreSQL etc.

In the other hand NoSQL databases are document-oriented, in this way articles, photographs, recordings, or a blog entry can be put into a single document. NoSQL databases offer another significant advantage, especially to application developers: simple entry. Relational databases have a full association with applications written in object-oriented programming languages like Java, PHP, and Python. NoSQL databases are frequently ready to evade this issue through APIs, which enable developers to execute inquiries without learning SQL or comprehend the fundamental engineering of their database framework. Some of the popular NoSQL databases are MongoDB, Apache's CouchDB, HBase.

4. SQL Advantages and Disadvantages:

Firstly, Considering the advantage of using a SQL database is because of its ACID properties. It offers high transaction reliability. Secondly it uses tables to store the data and the data is usually structured, that means it can store only one kind of data within a table. In addition to this SQL database guarantee crash recovery using crash recovery manager. Finally, Data confidentiality can be achieved only in Relational database system.

In the other hand, there are certain downfalls of using SQL databases, first downfall is that it does not suited for the cloud environments. Handling huge amount of data is also a big issue in SQL database.

5. No-SQL Advantages and Disadvantages:

Most of the restrictions which are mentioned above are resolved in NoSQL, these kinds of database are suitable if there is a requirement of storing large volumes of data and the data is unstructured. It also can be good option to use if the is a Quick advancement in an organisation. Usage of cloud is also a perk of using NoSQL database.

One of the limitations of NoSQL is lack of secure client communication, which is required for most of the companies these days. No-SQL is also does not support data warehousing, which is provided in SQL database. Most of the NoSQL database doesn't come with authentication and authorisation mechanism by default.

6. Recommendation

- In this research we compared SQL database with NoSQL database as two approaches for rebuilding the timetable for public transport Victoria.
- Though SQL database are better in structuring data and are more secure, it is not suitable for the organisations which has rapid development.
- SQL database can be used to the companies to ensure the ACID properties.
- But from the given description the usage of both website and app of public transport Victoria (PTV) has been growing and expected to grow in coming years, which requires huge amount of data, for storing the information.
- However, Considering the current scenario, we need a database which can store more data and should integrate well with both website and mobile app.
- In this case, NoSQL database server will be more futureproof and also be more reliable.

7.Conclusion

In conclusion, based on the above facts, I assume NoSQL will be perfect approach for rebuilding public transport timetable.

References

Rachit Agarwal. [Blog] Available at: https://www.algoworks.com/blog/sql-or-nosql-which-is-better-database-for-app-development/ [Accessed 11 Oct. 2018].

Carey Wodehouse. (2018). *SQL vs. NoSQL: What's the* difference? [online] Hiring | Upwork. Available at: https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/ [Accessed 11 Oct. 2018].