Database Concepts – Assignment 2

StudentID: s3828461

Part A:

1. **Functional Dependencies:**

Branch Table:

FD1: BranchNo -> B_Street, B_Suburb, B_Postcode, ManagerStaffNo

FD2: B_Suburb -> B_Postcode

FD3: BranchNo, ManagerStaffNo -> Start_Date, Monthly_Bonus, S_Name

PrimaryKey: BranchNo

This Branch table has multiple attributes or repeating groups of the same type. (Telephone1, Telephone2, Telephone3)
So we cannot state: BranchNo -> Telephone1, Telephone2, Telephone3

Staff Table:

FD4: StaffNo -> S_Name, S_Address, Position, Salary, Branch_No, SupervisorStaffNo

FD5: SupervisorStaffNo -> SupervisorStaffName

FD6: Position -> Salary

PrimaryKey: StaffNo

Job Table:

FD7: JobNo, CustNo -> Fault
FD8: JobNo, StaffNo -> Notes, Fee

Customer Table:

FD9: CustNo -> C_Street, C_Suburb, C_Postcode, C_Email

FD10: C_Suburb -> C_Postcode

PrimaryKey: CustNo

2. **Highest normal for each relation:**

Branch Table:

This is **not in 1NF**, because it has multiple attributes or repeating groups of the same type. (Telephone1, Telephone2, Telephone3)

Staff Table:

There are no repeating groups/attributes – 1NF
The primary key is single valued – 2NF

Highest normal form – 2NF

Not in 3NF as non-primary key attributes are functionally dependent on other non-primary key attributes as seen in FD5 and FD6.

Job Table:

There are no repeating groups/attributes – 1NF

Not in 2NF as not all non primary key attributes depend on the single primary key.

Customer Table:

There are no repeating groups/attributes – 1NF
The primary key is single valued – 2NF

Highest normal form – 2NF

Not in 3NF as non primary key attributes are functionally dependent on other non-primary key attributes as seen in FD10.

3. **Decomposing into 3NF Relations:**

Branch Table:

Branch (BranchNo, B_Street, B_Suburb, B_Postcode, ManagerStaffNo*, Start_Date, Monthly_Bonus, Telephone1, Telephone2, Telephone3)

Decomposing the multi valued Attribute:

Branch (BranchNo, B_Street, B_Suburb, B_Postcode, ManagerStaffNo*, Start_Date, Monthly_Bonus)

Branch_Telephone(BranchNo*, Telephone)

This is now in 3NF as no non primary key attribute is dependant on another non primary key attribute.

Decomposing to 3NF:

R1: Branch (BranchNo, B_Street, B_Suburb*,ManagerStaffNo*)

R2: Branch_Telephone(BranchNo*, Telephone)

R3: Branch_Manager(BranchNo*, ManagerStaffNo*, Start_Date, Monthly_Bonus, S_Name)

R4: Branch_Area(B_Suburb, B_Postcode)

Job Table:

Decomposing to 3NF:

Job(JobNo, CustNo*, Fault, StaffNo*, Notes, Fee)

The JobNo in the Job entity is a partial key, which means it cannot be a primary key on its own and it needs another column which is the foreign key of the owner entity. Therefore: as per FD7 AND FD8

R5: Job_customer(JobNo, CustNo*, Fault)

R6: Job_staff(JobNo, StaffNo*, Notes, Fee)

Staff Table:

Decomposing to 3NF:

R7: StaffNo(StaffNo, S_Name, S_Address, Position*, Branch_No, SupervisorStaffNo*)

R8: Staff_supervisor(SupervisorStaffNo, SupervisorStaffName)

R9: Staff_Position(Position , Salary)

Customer Table:

Decomposing to 3NF:

R10: Customer (CustNo, C_Street, C_Suburb*, C_Email)

R11: Customer_Destination(C_Suburb, C_Postcode)

4. **After Decomposing:**

R1: Branch (BranchNo, B_Street, B_Suburb*,ManagerStaffNo*)

R2: Branch_Telephone(BranchNo*, Telephone)

R3: Branch_Manager(BranchNo*, ManagerStaffNo*, Start_Date, Monthly_Bonus)

R4: Branch_Area(B_Suburb, B_Postcode)

R5: Job_customer(JobNo, CustNo*, Fault)

R6: Job_staff(JobNo, StaffNo*, Notes, Fee)

R7: StaffNo(StaffNo, S_Name, S_Address, Position*, Branch_No, SupervisorStaffNo*)

R8: Staff_supervisor(SupervisorStaffNo, SupervisorStaffName)

R9: Staff_Position(Position , Salary)

R10: Customer(CustNo, C_Street, C_Suburb*, C_Email)

R11: Customer_Designation(C_Suburb, C_Postcode)

From this final decomposition we can see that, both the relations R4 and R11 have the same primary key and primary key indicates uniqueness, therefore they can be combined.

R1: Branch (BranchNo, B_Street, B_Suburb*,ManagerStaffNo*)

R2: Branch_Telephone(BranchNo*, Telephone)

R3: Branch_Manager(BranchNo*, ManagerStaffNo*, Start_Date, Monthly_Bonus)

R4: Area(Suburb, Postcode)

R5: Job_customer(JobNo, CustNo*, Fault)

R6: Job_staff(JobNo, StaffNo*, Notes, Fee)

R7: StaffNo(StaffNo, S_Name, S_Address, Position*, Branch_No, SupervisorStaffNo*)

R8: Staff_supervisor(SupervisorStaffNo, SupervisorStaffName)

R9: Staff_Position(Position , Salary)

R10: Customer(CustNo, C_Street, C_Suburb*, C_Email)

5. **Final relational database schema:**

R1: Branch (BranchNo, B_Street, B_Suburb*,ManagerStaffNo*)

R2: Branch_Telephone(BranchNo*, Telephone)

R3: Branch_Manager(BranchNo*, ManagerStaffNo*, Start_Date, Monthly_Bonus)

R4: Area(Suburb, Postcode)

R5: Job_customer(JobNo, CustNo*, Fault)

R6: Job_staff(JobNo, StaffNo*, Notes, Fee)

R7: StaffNo(StaffNo, S_Name, S_Address, Position*, Branch_No, SupervisorStaffNo*)

R8: Staff_supervisor(SupervisorStaffNo, SupervisorStaffName)

R9: Staff_Position(Position , Salary)

R10: Customer(CustNo, C_Street, C_Suburb*, C_Email)

Part B: SQL

1.

```
SELECT * FROM BOOK WHERE YEAR > 1980;
```

2.a

```
SELECT firstname, lastname from author where authorID IN
(SELECT authorID from written_by where role = 'Translator' and bookdescID IN
(SELECT bookdescID from book where subjectID IN
(SELECT subjectID from subject where lower(subjecttype) = 'image processing')))
```

2.b

```
SELECT firstname, lastname from author
join written_by on author.authorID = written_by.authorID
join book on written_by.bookdescID = book.bookdescID
join subject on book.subjectID = subject.subjectID
where written_by.role = 'Translator' and subject.subjectID = (SELECT subjectID from
subject where lower(subjecttype) = 'image processing')
```

3.

```
SELECT firstname, middlename, lastname from author
join written_by on author.authorID = written_by.authorID
join book on written_by.bookdescID = book.bookdescID
where written_by.role = 'Author'
and lower(book.TITLE) = 'computing methods';
```

4.

```
select title from book
join book_copy on book.bookdescID = book_copy.bookdescID
where book_copy.BOOKID not in (select bookid from borrow_copy)
```

```
select title from book
join book_copy on book.bookdescID = book_copy.bookdescID
join borrow_copy on book_copy.bookID = borrow_copy.bookID
join borrow on borrow_copy.transactionID = borrow.transactionID
where borrow.borrowdate IS NOT NULL
```

5.a

```
select title from book
where lower(title) like '%computing'
or lower(title) like 'computing%'
or lower(title) like '%computing%'
```

5.b

```
select title from book
join written_by on written_by.bookdescID = book.bookdescID
join author on author.authorID = written_by.authorID
where written_by.role = 'Author'
and lower(book.title) = 'data processing concepts'
```

6.

```
Select distinct publisherfullname from publisher
Join published_by on publisher.publisherID = published_by.publisherID
Join Book on published_by.bookdescID = book.bookdescID
Where book.SUBJECTID = (select subjectid from subject where subjecttype = 'Computer
Software')
```

7.a

```
select firstname, middlename, lastname from author
left outer join written_by on author.authorID = written_by.authorID
left outer join book on written_by.bookdescID = book.bookdescID
where written_by.role = 'Author' or written_by.role = 'Translator' and
written_by.authorID is NULL
```

7.b

```
select firstname, middlename, lastname from author
where authorID not in (select authorID from
written_by where role = 'Translator' or role = 'Author'
and authorID is NULL
and bookdescID not in (select bookdescID from book))
```

8

```
select PUBLISHERFULLNAME from publisher where
exists (select PUBLISHERID from author join written_by using (authorid)
join published_by using(bookdescID)
join publisher using(publisherID)
where author.firstname != "ALFRED" and author.LASTNAME != "AHO");
```

9.

```
select firstname, lastname , count(book.BOOKDESCID) as 'bookswritten' from author
Join written_by on author.authorID = written_by.authorID
Join book on written_by.bookdescID = book.bookdescID
group by book.BOOKDESCID
having count(book.BOOKDESCID) > 3;
```

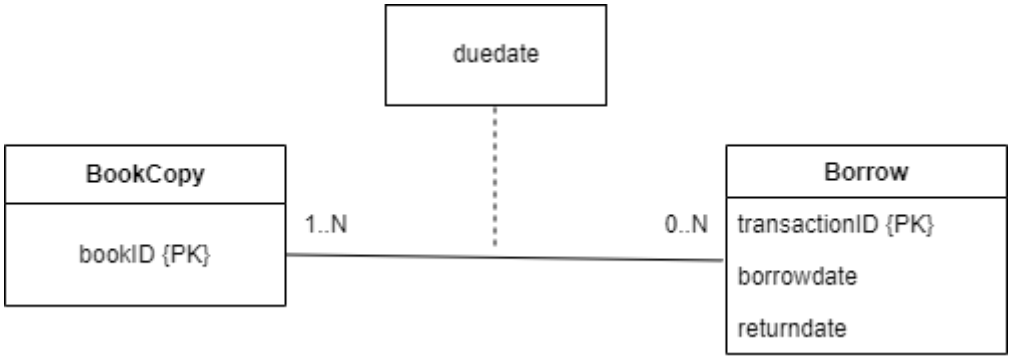
10.

```
select author.authorID,author.FIRSTNAME,author.LASTNAME,
count(written_by.BOOKDESCID) as bookswritten
from author join written_by
on author.authorID = written_by.authorID
join book on book.BOOKDESCID = written_by.BOOKDESCID
where role='Author'
group by written_by.authorID
having count(written_by.BOOKDESCID)=(select max(newtable.bookswritten) from author
join (select author.authorID, author.FIRSTNAME, author.LASTNAME,
count(written_by.BOOKDESCID) as bookswritten
from author join written_by
on author.authorID=written_by.authorID
join book on book.BOOKDESCID = written_by.BOOKDESCID
where role='Author' group by written_by.authorID )newtable)
```

11.

It is observed that all the books borrowed in one transaction are due at the same time and it should be returned at the same time. This is unreasonable as the books may be due on different days due to renew activity, and they can be returned separately no matter if they have a separate due date.

We can remove the due date attribute from the borrow entity and place it as a relational attribute between borrow and bookcopy.

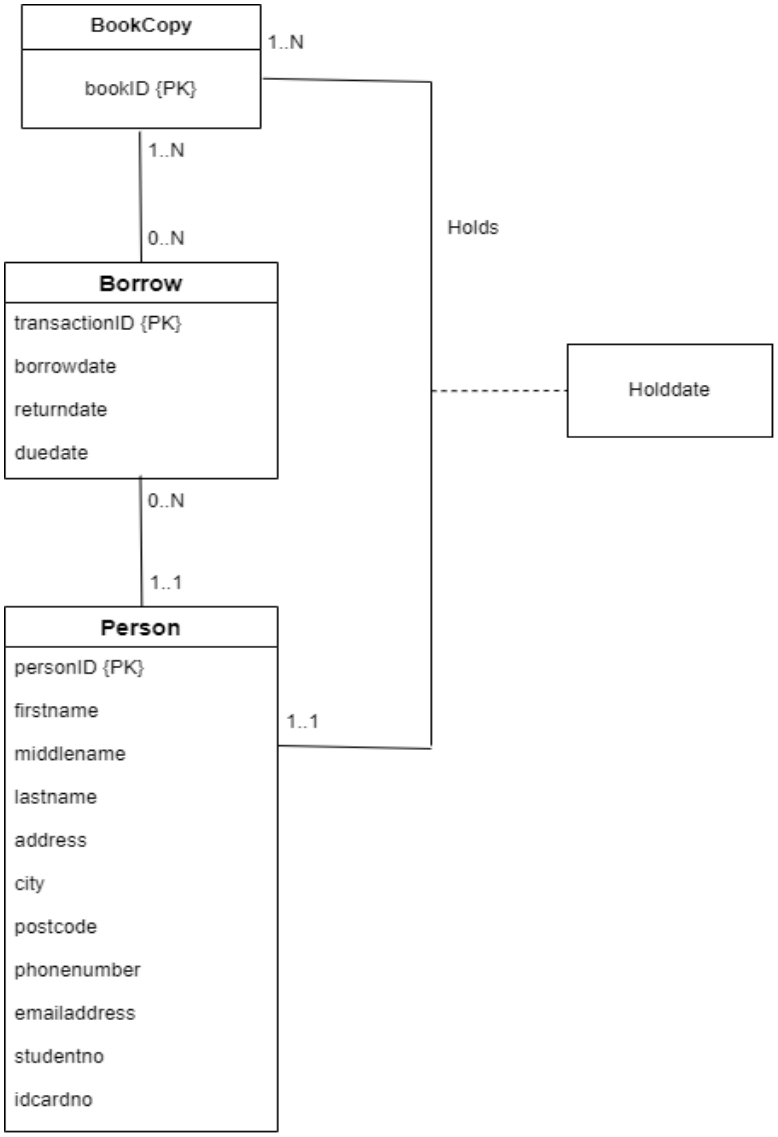


borrow(transactionID, personID*, borrowdate, returndate)

borrow_copy(transactionID*, bookID*, duedate)

12.

We can add a Holddate as a relational attribute between BookCopy and Person.

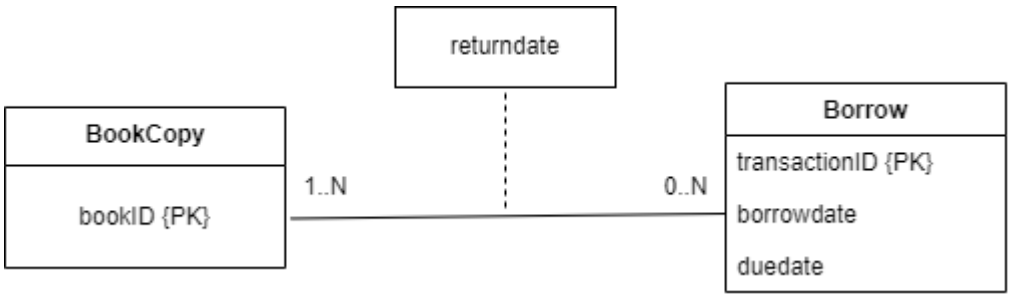


book_copy (bookID, transactionID*, personID*, Holddate)

13.

The library allows customers to return books in one transaction separately:

1. No the current database cannot support this operation and it requires modification.
2. The return date attribute can be removed from the Borrow entity and added as a relational attribute between BookCopy and Borrow, this way the database will support the operation.
3.



borrow(transactionID, personID*, borrowdate, duedate)

borrow_copy(transactionID*, bookID*, returndate)

Part C:

Jobactive Job Seeker is a digital free service app developed by the Department of Employment to help Australians find a job. This is a database which stores information such as Geo location, suburb name, post code, keyword and other job details like company, hours, title etc. On the whole, the system must be reliable, available and possess the ability to handle large volumes of data. There are several factors which need to be considered before choosing a backend database solution. Let us explore two options:

- **Traditional RDMS (such as Oracle and SQL Server):**

Relational database management system as the name suggests, it stores data and provides access to data points which is in a relation with each other. This is a very straightforward way to represent data, and data can be retrieved easily with the help of a single query – Structured Query Language (SQL). The form of data stored in the RDBMS is structured data. The information is organized in tables, with each row representing a record with a unique key value. The table's columns indicate attributes, and each record usually includes a value for each attribute, making it simple to see how data points are related.

In our case, the jobactive Job Seeker application, stores structured data such as Location, Company, Hours, Job title, Job no: etc. Therefore, SQL can easily be used to retrieve the following/ desired data. This makes it very easy to interact with management systems and it allows the joining of tables in the database with a few lines of code.

Our database follows certain **integrity** criteria to ensure that data is always correct and accessible. It uses the **atomicity** idea, which is essential for maintaining database data accurate and consistent with all rules and constraints. The **isolation** property ensures transaction is invisible until it is committed, and the **durability** attribute ensures that the

data changes are permanent once committed. The **consistency** defines the rules for maintaining the data points after a specific transaction.

Examples of RDBMS: MySQL, PostgreSQL, Oracle

Benefits:

Security: In our application, where the database structure will not be changed frequently, we can employ high level of security for the database.

Flexibility: DDL – Data definition language, this allows us to add new columns, tables, rename relations in the database and make alterations to the database schema.

Reduce Redundancy: Normalization is one of the main features of RDBMS. This is basically the practice of separating the data. For example: in a Customer and order scenario, thee customer details appear only once in a single entry and the order table stores a key which links to the customer table.

Ease of backup: Consistency is maintained throughout the database system which means the RDBMS are transactional. Back up is also made easy as most of the RDBMS offer import and export options. Cloud based RDBMS can be continuously monitored.

Drawbacks:

- A lot of care needs to be taken when designing the database schema initially.
- Requires skilled human resources to implement.
- A few RDBMS systems are expensive.
- Users may feel uncomfortable because the database tends to have a complex interface

- **NoSQL Database systems (such as MongoDB)**

Non relational data bases systems, store data in a non-tabular form. In this way they are totally different from traditional relational database systems. They store unstructured or semi-structured data and the databases might be based on data structures like json documents or often in key-value pairs. Most of the non-SQL databases allow horizontal scaling which mean we can add cheaper and commodity servers whenever there is a need.

These non RDBMS systems perform much faster because there is no query required to view the tables in order to provide the desired result. We can save huge amounts of time once we go schemeless. For application where the data might be changed frequently or handling different kinds or data, non RDBMS is an ideal solution.

They excel in possessing characteristics such as **availability, resilience, scalability, and the ease-of-use**. Thereby, they have the ability to support rapidly developing applications requiring to accommodate large amount of complex and unstructured data.

Examples of Non RDBMS: MongoDB, Redis, JanusGraph

Benefits:

Flexible schemas: They do not have a predefined set of rows and columns. They are structured more like a document.

Massive dataset organization: Can store large volumes of data and query them with ease. This is vital in the age of Big Data.

Multiple Data Structures: Whichever format the information is in, the non RDBMS can collate different type of information together in the same document.

Cloud: The cloud advantage plays a major role as the non RDBMS can grow exponentially and they will require a hosting environment to grow, expand and store them.

Drawbacks:

- The NoSQL databases do not offer properties such as atomicity, consistency, integrity and durability.
- There are a lot of NoSQL databases, but there is a lack of standardization in them. This is not the case in traditional RDBMS.
- The NoSQL database data models are predominantly optimized for queries and not for data duplication, the NoSQL Databases tend to be larger than the traditional RDBMS.
- Our application stores sensitive data such as user and job details, which can lead to a data leak. Since most of the web applications have experienced data leak, there is a lack of security.

- **Analysis and Conclusion:**

As stated above, we can see that the traditional RDBMS and the non-RDBMS systems have its own advantages and disadvantages. Before actually deciding on what database choose for the application, we should keep in mind the applications requirements. In our case, the jobactive Job Seeker Application requires the storage of job details such as Location, company, job No:, employment type etc.

Considering a traditional RDBMS, we possess the advantage of high level of security and prevention of data leakage. This will also help us to reduce redundancy and make the system more flexible overall. The SQL databases possess properties such as atomicity, consistency, integrity and durability. Normalization is also one of the main features of the standard SQL database which protects the data, eliminates redundancy and inconsistent dependencies. Therefore, for storing structural data this is the one-way solution.

But in a traditional RDBMS, it fails on a large note when it must deal with dynamic data.

Considering a non-RDBMS or a NoSQL database, it is the best solution when the application must store unstructured data in a dynamic manner. These databases work much faster than the traditional RDBMS which means the application development speed is much larger with NoSQL database. Therefore, for storing unstructured data NoSQL is the one-way solution.

But in a non-RDBMS, the security is a major issue as our jobactive application stores lots of sensitive data which cannot afford data leakage. Unlike traditional RDBMS, it lacks standardization and consistency.

Why SQL database for jobactive application?

In our case, the jobactive application stores the jobNo: which cannot be NULL. JobNo: is basically the unique number allotted to each job title. For validating the data, it is preferred we have schema so that we can also perform further analysis with ease.

The data for our application is structured data and it is best to be stored in a traditional RDBMS.

An approach for organizing data in a database is normalization. In SQL, the database normalized to avoid redundancy (data duplication) and ensure that only related data is stored in each table. This also prevents any problems caused by database changes like insertions, removals, and updates.

Due to extra SQL benefits such as transactions, SQL databases are often quick, dependable, and resilient.

Therefore, as a database expert I would conclude by recommending the usage of a traditional relational database management system/ SQL database as the backend database solution for the jobactive Job Seeker application rather than a non-relational database management system/ NoSQL database.

- **Future Scope:**

In a few ways, the NoSQL databases are more flexible than the traditional RDBMS. There are some scenarios where SQL databases can be improved.

1. The SQL database fails when it has to handle dynamic data.
2. The performance also deteriorates when it must handle large volumes of data.
3. Certain SQL databases are very expensive which is a major hinderance.

Ways to improve:

1. Implement horizontal scaling to SQL/ RDBMS
2. Reduce cost

REFERENCES:

Oracle.com. 2022. *What is a relational database?*. [online] Available at: <https://www.oracle.com/au/database/what-is-a-relational-database/> [Accessed 7 May 2022].

Education, I., 2022. *relational-databases*. [online] Ibm.com. Available at: <https://www.ibm.com/cloud/learn/relational-databases> [Accessed 7 May 2022].

Mongodb.com, 2022. *What is a non-relational database?*. [online] Mongodb.com. Available at: <https://www.mongodb.com/databases/non-relational > [Accessed 7 May 2022].

Mongodb.com, 2022. *Relational vs. Non-Relational Database?*. [online] Mongodb.com. Available at: <https://www.mongodb.com/databases/non-relational > [Accessed 14 May 2022].

microsoft.com, 2022. *Relational vs. NoSQL data*. [online] microsoft.com. Available at: <https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data> [Accessed 14 May 2022].