A cartoon of a coat of arms

AI-generated content may be incorrect.

TASK 1

Sophie Fidan 21068639

UFCFU3-15-3 Advanced

Databases

Table of Contents

[First Normal Form (1NF) 1](#_Toc190710742)

[Second Normal Form (2NF) 2](#_Toc190710743)

[Person 2](#_Toc190710744)

[Address 3](#_Toc190710745)

[Person-Address 4](#_Toc190710746)

[Favourite 5](#_Toc190710747)

[Neighbour 6](#_Toc190710748)

[Person-Neighbour 6](#_Toc190710749)

[Third Normal Form (3NF) 8](#_Toc190710750)

[Entity Relationship Diagram 8](#_Toc190710751)

[SQL Queries 9](#_Toc190710752)

[Query 1: Display all persons’ name and their ages in years 9](#_Toc190710753)

[Query 2: Group Persons by their favourite drink and return average age of each group 10](#_Toc190710754)

[Query 3: Display the average age of people who like Hiking 11](#_Toc190710755)

[Query 4: Display the total number of people from each City and sort it in ascending order by total number of people 11](#_Toc190710756)

[Query 5: Display name of person(s) whose neighbour is neighbour C 12](#_Toc190710757)

[Appendices 13](#_Toc190710758)

[A) Benchmark of Query 1 13](#_Toc190710759)

[B) Benchmark of Query 2 13](#_Toc190710760)

[C) Benchmark of Query 3 14](#_Toc190710761)

[D) Benchmark of Query 4 14](#_Toc190710762)

[E) Benchmark of Query 5 15](#_Toc190710763)

|  |
| --- |
| First Normal Form (1NF) |

In the First Normal Form, each column should contain an atomic value, and each record should be unique. Additionally, each row should have a unique identifier (primary key). For this purpose, a separate row is created for each neighbour of the person. The *person\_id* is created to ensure each row in the table is uniquely identifiable, even if other columns contain duplicate values. This *person\_id* is automatically incremented for each new entry, providing a unique identifier for each row.

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| Second Normal Form (2NF) |

In the Second Normal Form, all partial dependencies should be removed so that every non-key attribute should depend on the primary key. For this purpose, separate tables are created as *Person*, *Address*, *Favourite*, and *NeighboursPair*.

A diagram of a person

AI-generated content may be incorrect.

## Person

The *Person* table only contains information about the person such as *name*, *email* and *dob*(date of birth) with a primary key of *person\_id.*

## Address

The address attributes (*street*, city, *country* and *zipcode*) were separated into a new table called *Address* with a primary key of a*ddress\_id*.

The *Address* table has many-to-many relationships to the *Person* table with a foreign key of *person\_id,* meaning multiple addresses can belong to one person, and two different people can share the same address. Even though the use case did not include any example of two people sharing the same address, this design enhances the flexibility of the database.

## NeighboursPair

The *NeighboursPair* table contains the neighbour information for each address, including their names and emails as *neighbour1\_name*, *neighbour2\_name*, *neighbour1\_email*, and *neighbour2\_email*.

This table has a one-to-one relationship with the *Address* table, which means every address can exactly have one neighbour pair (two neighbours) associated with it. The relationship is indicated with *address\_id* as a primary key of the *NeighboursPair* table as well as a foreign key to the *Address* table.

## Favourite

The *Favourite* table consists of the user’s favourite information. This table is created with attributes of *favoruite\_id* as the primary key, *type* and *value*. The type is the type of the favourite, which in this case, is either drink, activity or book. The value is the value of this type, which is the *favourite\_book*, *favoruite\_drink*, and *favourite\_activity.*

There are many-to-one relationships with the Person table, meaning multiple favourites can belong to one person. This relationship is represented with *person\_id* as a foreign key to the *Person* table in the diagram.

**Design Rationale**

The Favourite attributes () could have been placed in a *Favourite* table with separate columns for each type of favourite and still be in 2NF. However, creating a *Favourite* table with *type* and *value* columns reduces the database design cost and enhances the scalability of the database design for the following reasons:

* This type-value structure uniforms the data model. Therefore, the system can accommodate new types of favourites without altering the table schema, providing more dynamic expansion. For instance, adding a new favourite type called favourite destination only requires a new row in the table.
* This design simplifies queries to retrieve favourite information. Instead of querying multiple columns for different favourite types, the system can query a single table and filter by type.

If there is any additional information about the book, activity, or drink (such as the author of the book, or the drink’s flavour), then the creation of separate tables for each type would be necessary. However, the current use case did not provide any such information, making the type-value structure a more suitable choice.

|  |
| --- |
| Third Normal Form (3NF) |

In the Third Normal Form, all the attributes must be functionally dependent on the primary key, and there should be no transitive dependency, which means non-key attributes should not be dependent on other non-key attributes. Therefore, the associated entities that have many-to-many relationships should be further normalised.

## Person

By fulfilling the 2NF, the Person table already satisfies the requirements of 3NF.

## PersonAddress

The *PersonAddress* serves as a pivotal junction table that resolves the many-to-many relationship by establishing a many-to-one relationship with the *Person* table and a many-to-one relationship with the *Address* table. This means that multiple entries in the *PersonAddress* table can reference the same person and multiple entries in the *PersonAddress* can reference the same address.

**Composite Primary Key**

The *PersonAddress* table uses a composite primary key, which is a combination of *person\_id* and *address\_id*. This composite key uniquely identifies each entry in the *PersonAddress* table, ensuring that each combination of person and address is unique. This design eliminates redundancy by preventing duplicate entries for the same person-address pair.

In this refined schema, each person can be associated with multiple addresses, and each address can be linked to multiple persons. This fulfils the requirements of 3NF, ensuring there are no transitive dependencies, thereby maintaining the data integrity and reduce data redundancy.

## Address

Since the *PersonAddress* table is used to manage the many-to-many relationship between *Person* and *Address*, the foreign key *person\_id* in the *Address* table is redundant and can be removed. This is because the *PersonAddress* table already captures the relationship between persons and addresses.

## Favourite

By fulfilling the 2NF, the Person table already satisfies the requirements of 3NF.

|  |
| --- |
| Entity Relationship Diagram |

The Entity Relationship diagram is created using PlantUML as below:

A diagram of a person

AI-generated content may be incorrect.

|  |
| --- |
| SQL Queries |

To make SQL queries more efficient, unique indexes are created:

CREATE UNIQUE INDEX person\_id\_index ON Person (id);

CREATE UNIQUE INDEX address\_id\_index ON Address (id);

CREATE UNIQUE INDEX neighbour\_id\_index ON Neighbour (id);

CREATE UNIQUE INDEX person\_address\_id\_index ON PersonAddress (id);

CREATE UNIQUE INDEX person\_neighbour\_id\_index ON PersonNeighbour (id);

CREATE UNIQUE INDEX favourite\_id\_index ON Favourite (id);

## Query 1: Display all persons’ name and their ages in years

SELECT name,

TIMESTAMPDIFF(YEAR, dob, CURDATE()) AS age

FROM Person;

A screenshot of a black screen

AI-generated content may be incorrect.

The benchmark of Query 1 before and after adding a unique index can be found in [Appendix A](#_Benchmark_of_Query).

## Query 2: Group Persons by their favourite drink and return average age of each group

SELECT f.value AS favourite\_drink,

ROUND(AVG(TIMESTAMPDIFF(YEAR, p.dob, CURDATE())), 2) AS average\_age

FROM Person p

JOIN Favourite f ON p.id = f.person\_id

WHERE f.type = 'Drink'

GROUP BY f.value;

A screenshot of a menu

AI-generated content may be incorrect.

The benchmark of Query 2 before and after adding a unique index can be found in [Appendix B](#_Benchmark_of_Query_1).

## Query 3: Display the average age of people who like Hiking

SELECT f.value as activity, ROUND(AVG(TIMESTAMPDIFF(YEAR, dob, CURDATE())), 2) AS average\_age

FROM Person p

JOIN Favourite f ON p.id = f.person\_id

WHERE f.type = 'Activity' AND f.value = 'Hiking';

A screenshot of a computer

AI-generated content may be incorrect.

The benchmark of Query 3 before and after adding a unique index can be found in [Appendix C](#_Benchmark_of_Query_2).

## Query 4: Display the total number of people from each City and sort it in ascending order by total number of people

SELECT a.city, COUNT(pa.person\_id) AS total\_number\_of\_people

FROM Address a

JOIN PersonAddress pa ON a.id = pa.address\_id

GROUP BY a.city

ORDER BY total\_number\_of\_people ASC;

A screenshot of a black screen

AI-generated content may be incorrect.

The benchmark of Query 4 before and after adding a unique index can be found in [Appendix D](#_Benchmark_of_Query_3).

## Query 5: Display name of person(s) whose neighbour is neighbour C

SELECT p.name AS person\_name, n.name AS neighbour\_name

FROM Person p

JOIN PersonNeighbour pn ON p.id = pn.person\_id

JOIN Neighbour n ON pn.neighbour\_id = n.id

WHERE n.name = 'Neighbor C';

A screenshot of a phone

AI-generated content may be incorrect.

The benchmark of Query 5 before and after adding a unique index can be found in [Appendix E](#_Benchmark_of_Query_4).

|  |
| --- |
| Appendices |

## Benchmark of Query 1

Display all persons’ name and their ages in years

A black background with white text

AI-generated content may be incorrect.

Figure 1: The benchmark of Query 1 without indexing

A black and white text

AI-generated content may be incorrect.

Figure 2: The benchmark of Query 1 after creating unique indexes

## Benchmark of Query 2

Group Persons by their favourite drink and return average age of each group

A computer screen with text

AI-generated content may be incorrect.

Figure 3: The benchmark of Query 2 without indexing

A computer screen with text

AI-generated content may be incorrect.

Figure 4: The benchmark of Query 2 after creating unique indexes

## Benchmark of Query 3

Display average age of people who likes Hiking

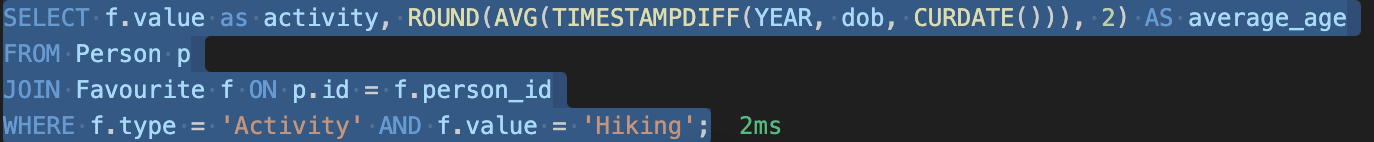


Figure 5: The benchmark of Query 3 without indexing

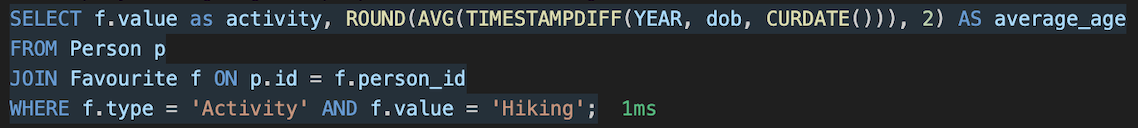


Figure 6: The benchmark of Query 3 after creating unique indexes

## Benchmark of Query 4

Display the total number of people from each City and sort it in ascending order by total number of people

A computer screen with text

AI-generated content may be incorrect.

Figure 7: The benchmark of Query 4 without indexing

A black background with white text

AI-generated content may be incorrect.

Figure 8: The benchmark of Query 4 after creating unique indexes

## Benchmark of Query 5

Display name of person(s) whose neighbour is neighbour C

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 9: The benchmark of Query 5 without indexing

A computer screen with text

AI-generated content may be incorrect.

Figure 10: The benchmark of Query 5 after creating unique indexes