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) |

A screenshot of a computer screen

AI-generated content may be incorrect.A white box with black text

AI-generated content may be incorrect.In 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 *id* is created to ensure each row in the table is uniquely identifiable, even if other columns contain duplicate values. This *id* is automatically incremented for each new entry, providing a unique identifier for each row.

CREATE TABLE `1NF` (

`id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`email` varchar(255) NOT NULL,

`dob` date NOT NULL,

`street` varchar(255) NOT NULL,

`city` varchar(255) NOT NULL,

`country` varchar(255) NOT NULL,

`zipcode` varchar(20) NOT NULL,

`favourite\_book` varchar(255) NOT NULL,

`favourite\_drink` varchar(255) NOT NULL,

`favourite\_activity` varchar(255) NOT NULL,

`neighbour\_name` varchar(255) NOT NULL,

`neighbour\_email` varchar(255) NOT NULL,

PRIMARY KEY (`id`)

)

|  |
| --- |
| 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*, *PersonAddress, Favourite*, *Neighbour* and *PersonNeighbour*.

## Person

The Person table only contains information about the person:

CREATE TABLE `Person` (

`id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`email` varchar(255) NOT NULL,

`dob` date NOT NULL,

PRIMARY KEY (`id`)

)

A screenshot of a phone

AI-generated content may be incorrect.

## Address

The address attributes (*street*, city, *country* and *zipcode*) were separated into a new table called *Address* as well as the *type* of address to enable the system to add different types of addresses such as home, work, etc. Even though the given use case did not include any example of different types of address, this design enhances the flexibility and scalability of the database.

CREATE TABLE `Address` (

`id` int NOT NULL AUTO\_INCREMENT,

`type` varchar(255) DEFAULT NULL,

`street` varchar(255) NOT NULL,

`city` varchar(255) NOT NULL,

`country` varchar(255) NOT NULL,

`zipcode` varchar(20) NOT NULL,

PRIMARY KEY (`id`)

)

A screenshot of a computer

AI-generated content may be incorrect.

## Person-Address

The *PersonAddress* table is a junction table that establishes a many-to-many relationship between the *Person* and *Address* entities to eliminate partial dependency. Each row in the *PersonAddress* table represents a unique relationship between a person and their addresses. The *Address* table could have included person\_id as a foreign key to the *Person* table, but this would imply that each person can have only one address information for each type of address. For instance, each person can have only one home address. This design ensures that a person can have multiple home addresses, multiple work addresses or any type of choice.

A screenshot of a black and white screen

AI-generated content may be incorrect.

## Favourite

The *Favourite* table only consists of the user’s favourite information. This table is created with attributes as *id* for primary key, *type*, value, and *person\_id* as a foreign key to the Person table. The Favourite attributes (favourite\_books, *favoruite\_drink*, *favourite\_activity*) 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 enhances the flexibility and scalability of the database design:

* 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.

CREATE TABLE `Favourite` (

`id` int NOT NULL AUTO\_INCREMENT,

`type` varchar(255) NOT NULL,

`value` varchar(255) NOT NULL,

`person\_id` int DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `person\_id` (`person\_id`),

CONSTRAINT `Favourite\_ibfk\_1` FOREIGN KEY (`person\_id`) REFERENCES `Person` (`id`)

)

A screenshot of a black screen

AI-generated content may be incorrect.

## Neighbour

The Neighbour table contains the neighbour information:

CREATE TABLE `Neighbour` (

`id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(255) NOT NULL,

`email` varchar(255) NOT NULL,

PRIMARY KEY (`id`)

)

A screenshot of a black screen

AI-generated content may be incorrect.

## Person-Neighbour

The *PersonNeighbour* table is a junction table that establishes a many-to-many relationship between the *Person* and *Neighbour* entities to eliminate partial dependency. Each row in the *PersonNeighbour* table represents a unique relationship between a person and their neighbour. The *Neighbour* table could have included person\_id as a foreign key to the *Person* table, but this would imply that each neighbour is associated with only one person, or their information would need to be repeated for each person, leading to data duplication. This does not accurately represent the real-world scenario where a neighbour can also be another person’s neighbour even though the use case did not include any example. This design minimises data redundancy and maintains data integrity properly.

CREATE TABLE `PersonNeighbour`

`id` int NOT NULL AUTO\_INCREMENT,

`person\_id` int DEFAULT NULL,

`neighbour\_id` int DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `person\_id` (`person\_id`),

KEY `neighbour\_id` (`neighbour\_id`),

CONSTRAINT `PersonNeighbour\_ibfk\_1` FOREIGN KEY (`person\_id`) REFERENCES `Person` (`id`),

CONSTRAINT `PersonNeighbour\_ibfk\_2` FOREIGN KEY (`neighbour\_id`) REFERENCES `Neighbour` (`id`)

)

A screenshot of a black wall

AI-generated content may be incorrect.

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

In 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. By fulfilling the 2NF for each table, the design already satisfies the requirements of 3NF.

* The *Person* table has a one-to-many relationship with the *Favourite* table, represented by the foreign key *person\_id* in the Favourite table, which means the one person can have 0 or more favourites.
* The *Person* table has a many-to-many relationship with the *Neighbour* table, represented by *PersonNeighbour* junction table, which means one person can have 0 or more neighbours.
* The *Person* table has a many-to-many relationship with the *Address* table, represented by *PersonAddress* junction table, which means one person can have 0 or more addresses.

|  |
| --- |
| 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

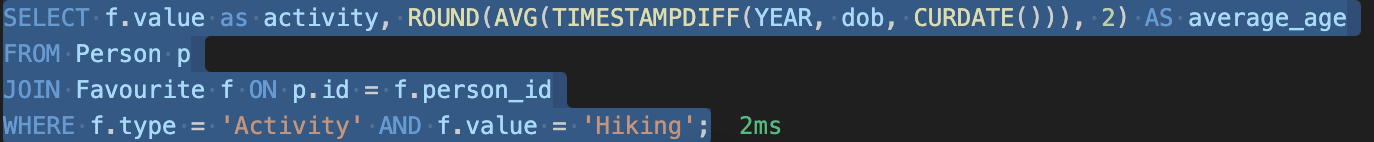


Figure 5: The benchmark of Query 3 without indexing

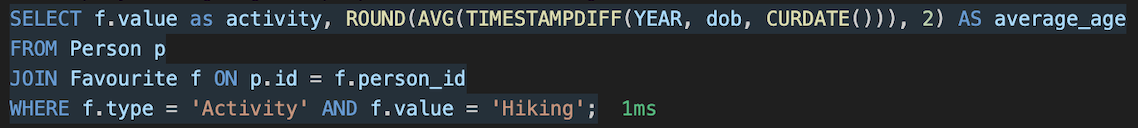


Figure 6: The benchmark of Query 3 after creating unique indexes

## 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