TASK 1

Sophie Fidan
21068639

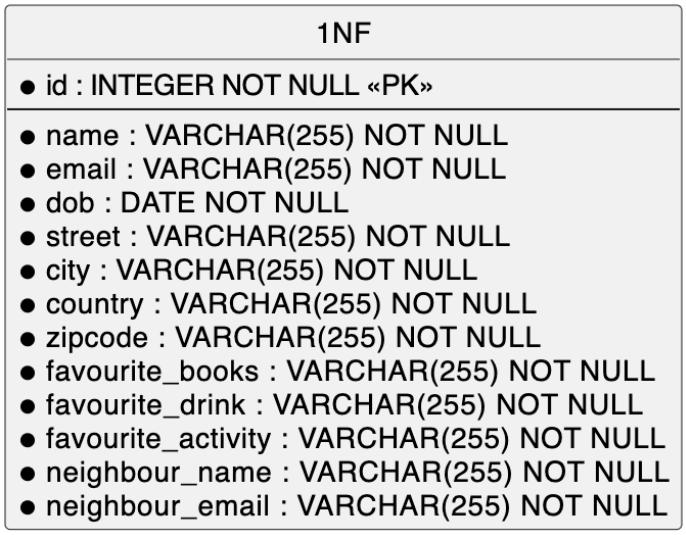
UFCFU3-15-3
Advanced

Databases

TABLE OF CONTENTS

FIRST NORMAL FORM (1NF)

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.




```sql
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`)
)
```

id int	name varchar(255)	email varchar(255)	dob date	street varchar(255)	city varchar(255)	country varchar(255)	zipcode varchar(20)	favourite_book varchar(255)	favourite_drink varchar(255)	favourite_activity varchar(255)	neighbour_name varchar(255)	neighbour_email varchar(255)
1	Person 1	person1@email.com	1995-03-15	12 Maple St	London	England	E1 6AN	A New Beginning	Lemonade	Outdoor Running	Neighbor A	neighborA@email.com
2	Person 1	person1@email.com	1995-03-16	13 Maple St	London	England	E1 6AN	A New Beginning	Lemonade	Outdoor Running	Neighbor B	neighborB@email.com
3	Person 2	person2@email.com	1993-06-22	45 Oak Ave	Manchester	England	M1 2WD	The Road to Success	Coffee	Hiking	Neighbor C	neighborC@email.com
4	Person 2	person2@email.com	1993-06-22	45 Oak Ave	Manchester	England	M1 2WD	The Road to Success	Coffee	Hiking	Neighbor D	neighborD@email.com
5	Person 3	person3@email.com	1991-09-10	89 Pine Rd	Birmingham	England	B1 1AB	Endless Possibilities	Smoothie	Swimming	Neighbor E	neighborE@email.com
6	Person 4	person3@email.com	1991-09-11	90 Pine Rd	Birmingham	England	B1 1AB	Endless Possibilities	Smoothie	Swimming	Neighbor F	neighborF@email.com
7	Person 4	person4@email.com	1998-12-05	23 Birch St	Edinburgh	Scotland	EH1 1YZ	Journey of Life	Iced Tea	Traveling	Neighbor G	neighborG@email.com
8	Person 4	person4@email.com	1998-12-05	23 Birch St	Edinburgh	Scotland	EH1 1YZ	Journey of Life	Iced Tea	Traveling	Neighbor H	neighborH@email.com
9	Person 5	person5@email.com	1983-11-30	67 Cedar Ln	Bristol	England	BS1 3XE	The Adventure Continues	Green Tea	Gardening	Neighbor I	neighborI@email.com
10	Person 6	person5@email.com	1983-12-01	68 Cedar Ln	Bristol	England	BS1 3XE	The Adventure Continues	Green Tea	Gardening	Neighbor J	neighborJ@email.com
11	Person 6	person6@email.com	1989-07-18	56 Elm St	Liverpool	England	L1 1AA	Finding Inner Peace	Coconut Water	Reading	Neighbor K	neighborK@email.com
12	Person 6	person6@email.com	1989-07-18	56 Elm St	Liverpool	England	L1 1AA	Finding Inner Peace	Coconut Water	Reading	Neighbor L	neighborL@email.com
13	Person 7	person7@email.com	1996-04-25	12 Maple St	Glasgow	Scotland	G1 2TF	Exploring New Horizons	Fruit Juice	Cycling	Neighbor M	neighborM@email.com
14	Person 7	person7@email.com	1996-04-25	12 Maple St	Glasgow	Scotland	G1 2TF	Exploring New Horizons	Fruit Juice	Cycling	Neighbor N	neighborN@email.com
15	Person 8	person8@email.com	1990-01-09	89 Oak Dr	Leeds	England	LS1 3AB	The Great Journey	Water	Hiking	Neighbor O	neighborO@email.com
16	Person 8	person8@email.com	1990-01-09	89 Oak Dr	Leeds	England	LS1 3AB	The Great Journey	Water	Hiking	Neighbor P	neighborP@email.com
17	Person 9	person9@email.com	1993-08-17	123 Pine Rd	Newcastle	England	NE1 2AB	The Power of Change	Hot Chocolate	Skiing	Neighbor Q	neighborQ@email.com
18	Person 9	person9@email.com	1993-08-17	123 Pine Rd	Newcastle	England	NE1 2AB	The Power of Change	Hot Chocolate	Skiing	Neighbor R	neighborR@email.com
19	Person 10	person10@email.com	1997-10-22	15 Elm St	Cardiff	Wales	CF10 3AF	New Beginnings Await	Fruit Smoothie	Jogging	Neighbor S	neighborS@email.com
20	Person 11	person10@email.com	1997-10-23	16 Elm St	Cardiff	Wales	CF10 3AF	New Beginnings Await	Fruit Smoothie	Jogging	Neighbor T	neighborT@email.com
21	Person 11	person11@email.com	1992-05-13	78 Oak Ln	Sheffield	England	S1 4GT	Wandering Souls	Sparkling Water	Rock Climbing	Neighbor U	neighborU@email.com
22	Person 12	person11@email.com	1992-05-14	79 Oak Ln	Sheffield	England	S1 4GT	Wandering Souls	Sparkling Water	Rock Climbing	Neighbor V	neighborV@email.com
23	Person 12	person12@email.com	1986-02-27	56 Birch Rd	Nottingham	England	NG1 2PB	Freedom and Choice	Herbal Tea	Yoga	Neighbor W	neighborW@email.com
24	Person 12	person12@email.com	1986-02-27	56 Birch Rd	Nottingham	England	NG1 2PB	Freedom and Choice	Herbal Tea	Yoga	Neighbor X	neighborX@email.com
25	Person 13	person13@email.com	1991-11-25	10 Holy St	Cardiff	Wales	CF10 2NF	New Beginnings Await	Smoothie	Hiking	Neighbor Y	neighborY@email.com
26	Person 13	person13@email.com	1991-11-25	10 Holy St	Cardiff	Wales	CF10 2NF	New Beginnings Await	Smoothie	Hiking	Neighbor Z	neighborZ@email.com

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`)
)
```

* id int	* name varchar(255)	* email varchar(255)	* dob date
1	Person 1	person1@email.com	1995-03-15
2	Person 2	person2@email.com	1993-06-22
3	Person 3	person3@email.com	1991-09-10
4	Person 4	person4@email.com	1998-12-05
5	Person 5	person5@email.com	1983-11-30
6	Person 6	person6@email.com	1989-07-18
7	Person 7	person7@email.com	1996-04-25
8	Person 8	person8@email.com	1990-01-09
9	Person 9	person9@email.com	1993-08-17
10	Person 10	person10@email.com	1997-10-22
11	Person 11	person11@email.com	1992-05-13
12	Person 12	person12@email.com	1986-02-27
13	Person 13	person13@email.com	1991-11-25
14	Person 14	person14@email.com	1987-02-01
15	Person 15	person15@email.com	1984-08-12
16	Person 16	person16@email.com	1990-03-09
17	Person 17	person17@email.com	1995-11-17
18	Person 18	person18@email.com	1994-06-20
19	Person 19	person19@email.com	1992-12-11
20	Person 20	person20@email.com	1988-09-25

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`)
)
```

* id int	type varchar(255)	* street varchar(255)	* city varchar(255)	* country varchar(255)	* zipcode varchar(20)
1	Home	12 Maple St	London	England	E1 6AN
2	Home	45 Oak Ave	Manchester	England	M1 2WD
3	Home	89 Pine Rd	Birmingham	England	B1 1AB
4	Home	23 Birch St	Edinburgh	Scotland	EH1 1YZ
5	Home	67 Cedar Ln	Bristol	England	BS1 3XE
6	Home	56 Elm St	Liverpool	England	L1 1AA
7	Home	12 Maple St	Glasgow	Scotland	G1 2TF
8	Home	89 Oak Dr	Leeds	England	LS1 3AB
9	Home	123 Pine Rd	Newcastle	England	NE1 2AB
10	Home	15 Elm St	Cardiff	Wales	CF10 3AF
11	Home	78 Oak Ln	Sheffield	England	S1 4GT
12	Home	56 Birch Rd	Nottingham	England	NG1 2PB
13	Home	10 Holy St	Cardiff	Wales	CF10 2NF
14	Home	34 Willow Rd	Edinburgh	Scotland	EH1 1AB
15	Home	78 Cedar Ave	Cambridge	England	CB1 2SE
16	Home	45 Maple Rd	Oxford	England	OX2 6TP
17	Home	23 Birch Ave	Southampton	England	SO14 3HL
18	Home	12 Elm Blvd	Leicester	England	LE1 3PL
19	Home	56 Oak Rd	Norwich	England	NR1 4BE
20	Home	89 Pine Ave	Cardiff	Wales	CF10 3BC

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.

* id int	person_id int	address_id int
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20	20	20

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*, *favouriate_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`)
)
```

* id int	* type varchar(255)	* value varchar(255)	person_id int
1	Book	A New Beginning	1
2	Drink	Lemonade	1
3	Activity	Outdoor Running	1
4	Book	The Road to Success	2
5	Drink	Coffee	2
6	Activity	Hiking	2
7	Book	Endless Possibilities	3
8	Drink	Smoothie	3
9	Activity	Swimming	3
10	Book	Journey of Life	4
11	Drink	Iced Tea	4
12	Activity	Traveling	4
13	Book	The Adventure Continues	5
14	Drink	Green Tea	5
15	Activity	Gardening	5
16	Book	Finding Inner Peace	6
17	Drink	Coconut Water	6
18	Activity	Reading	6
19	Book	Exploring New Horizons	7
20	Drink	Fruit Juice	7
21	Activity	Cycling	7
22	Book	The Great Journey	8
23	Drink	Water	8
24	Activity	Hiking	8
25	Book	The Power of Change	9
26	Drink	Hot Chocolate	9

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`)
)
```

* id int	* name varchar(255)	* email varchar(255)
1	Neighbor A	neighborA@email.com
2	Neighbor B	neighborB@email.com
3	Neighbor C	neighborC@email.com
4	Neighbor D	neighborD@email.com
5	Neighbor E	neighborE@email.com
6	Neighbor F	neighborF@email.com
7	Neighbor G	neighborG@email.comthi
8	Neighbor H	neighborH@email.com
9	Neighbor I	neighborI@email.com
10	Neighbor J	neighborJ@email.com
11	Neighbor K	neighborK@email.com
12	Neighbor L	neighborL@email.com
13	Neighbor M	neighborM@email.com
14	Neighbor N	neighborN@email.com
15	Neighbor O	neighborO@email.com
16	Neighbor P	neighborP@email.com
17	Neighbor Q	neighborQ@email.com
18	Neighbor R	neighborR@email.com
19	Neighbor S	neighborS@email.com
20	Neighbor T	neighborT@email.com
21	Neighbor U	neighborU@email.com
22	Neighbor V	neighborV@email.com
23	Neighbor W	neighborW@email.com

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`)
)
```

* id int	person_id int	neighbour_id int
1	1	1
2	1	2
3	2	3
4	2	4
5	3	5
6	3	6
7	4	7
8	4	8
9	5	9
10	5	10
11	6	11
12	6	12
13	7	13
14	7	14
15	8	15
16	8	16
17	9	17
18	9	18
19	10	19
20	10	20
21	11	21
22	11	22
23	12	23

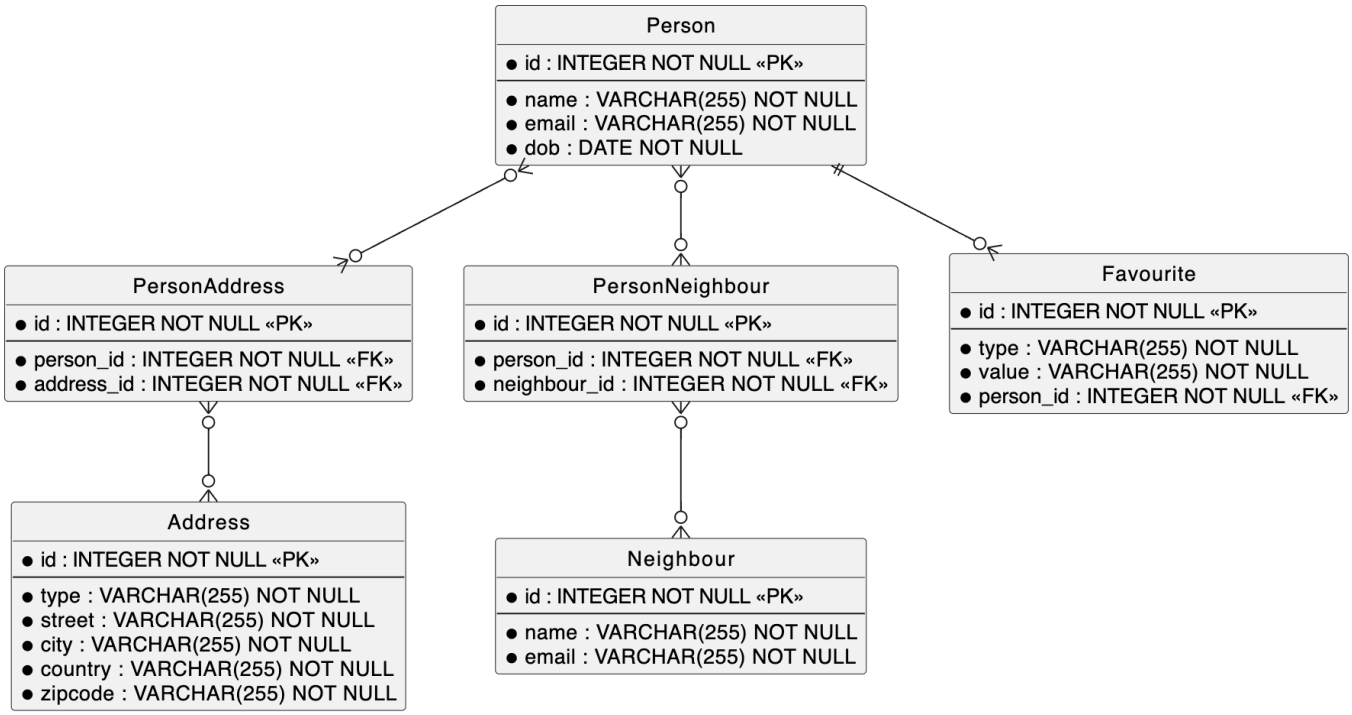
THIRD NORMAL FORM (3NF)

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:



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;
```

* name varchar(255)	age bigint
Person 1	29
Person 2	31
Person 3	33
Person 4	26
Person 5	41
Person 6	35
Person 7	28
Person 8	35
Person 9	31
Person 10	27
Person 11	32
Person 12	38
Person 13	33
Person 14	38
Person 15	40
Person 16	34
Person 17	29
Person 18	30
Person 19	32
Person 20	36

The benchmark of Query 1 before and after adding a unique index can be found in Appendix A.

QUERY 2: GROUP PERSONS BY THEIR FAVOURITE DRINK AND RETURN AVERAGE AGE OF EACH GROUP

```sql
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;
```

* favourite_drink varchar(255)	average_age decimal(23,2)
Lemonade	33.50
Coffee	31.00
Smoothie	33.00
Iced Tea	26.00
Green Tea	35.00
Coconut Water	32.50
Fruit Juice	31.00
Water	35.50
Hot Chocolate	31.00
Fruit Smoothie	27.00
Sparkling Water	32.00
Herbal Tea	35.00
Iced Coffee	40.00

The benchmark of Query 2 before and after adding a unique index can be found in Appendix B.

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';
```

activity varchar	average_age decimal
Hiking	33.00

The benchmark of Query 3 before and after adding a unique index can be found in Appendix C.

QUERY 4: DISPLAY THE TOTAL NUMBER OF PEOPLE FROM EACH CITY AND SORT IT IN ASCENDING ORDER BY TOTAL NUMBER OF PEOPLE

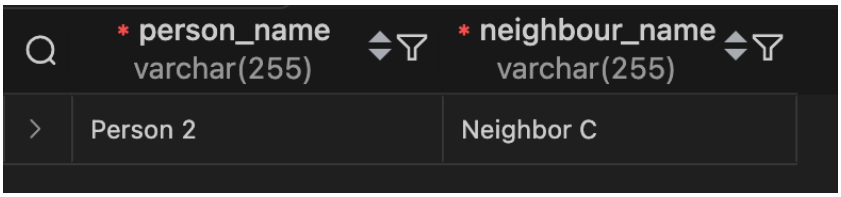
```
SELECT a.city, COUNT(pa.person_id) AS total_number_of_people
FROM Address a
JOIN PersonAddress pa ON a.id = pa.address_id
GROUP BY a.city
ORDER BY total_number_of_people ASC;
```

city varchar(255)	total_number_of_people bigint
Newcastle	1
Norwich	1
Leicester	1
Southampton	1
Oxford	1
Cambridge	1
Nottingham	1
Sheffield	1
London	1
Leeds	1
Glasgow	1
Liverpool	1
Bristol	1
Birmingham	1
Manchester	1
Edinburgh	2
Cardiff	3

The benchmark of Query 4 before and after adding a unique index can be found in Appendix D.

QUERY 5: DISPLAY NAME OF PERSON(S) WHOSE NEIGHBOUR IS NEIGHBOUR C

```sql
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';
```



* person_name varchar(255)	* neighbour_name varchar(255)
Person 2	Neighbor C

The benchmark of Query 5 before and after adding a unique index can be found in Appendix E.

A) BENCHMARK OF QUERY 1

Display all persons' name and their ages in years

```
SELECT name,
       TIMESTAMPDIFF(YEAR, dob, CURDATE()) AS age
FROM Person;  2ms
```

Figure 1: The benchmark of Query 1 without indexing

```
SELECT name,
       TIMESTAMPDIFF(YEAR, dob, CURDATE()) AS age
FROM Person;  1ms
```

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

B) BENCHMARK OF 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;  2ms
```

Figure 3: The benchmark of Query 2 without indexing

```
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;  1ms
```

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

C) BENCHMARK OF QUERY 3

Display average age of people who likes 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';  2ms
```

Figure 5: The benchmark of Query 3 without indexing

```
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';  1ms
```

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

D) BENCHMARK OF 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;  3ms
```

Figure 7: The benchmark of Query 4 without indexing

```
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;  1ms
```

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

E) BENCHMARK OF 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';  2ms
```

Figure 9: The benchmark of Query 5 without indexing

```
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';  1ms
```

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