CAiSE'24

JUNE 4, 2024
LIMASSOL, CYPRUS

EOMAS

workshop on
Enterprise **O**rganization **M**odeling **a**nd **S**imulation

CONCEPTUAL DATA NORMALISATION FROM THE PRACTICAL VIEW OF USING GRAPH DATABASES

VOJTĚCH MERUNKA, HIMESHA WIJEKOON, PAVEL BERÁNEK

CZECH UNIVERSITY OF LIFE SCIENCES, PRAGUE, CZECHIA

CONTENTS

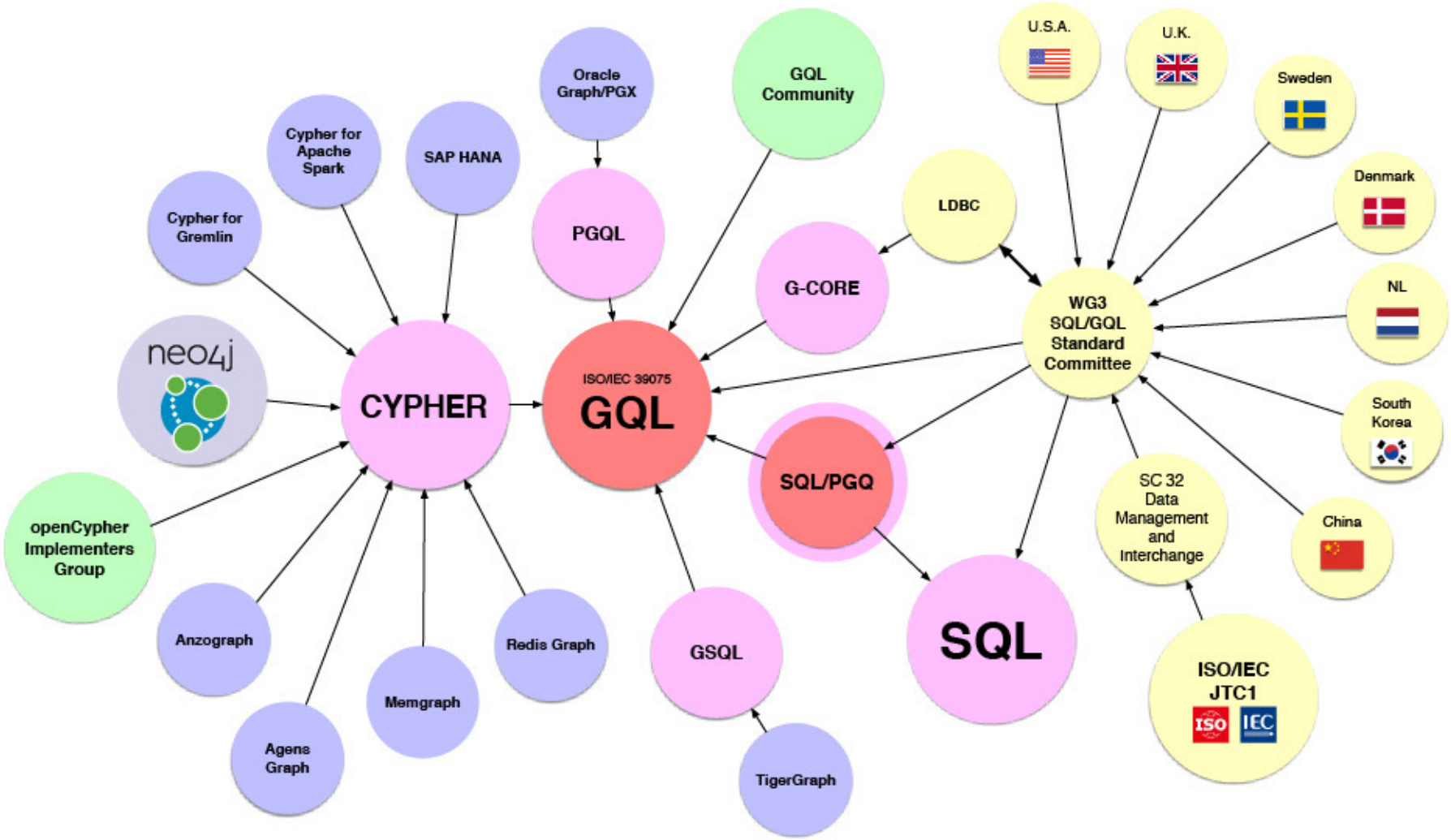
a) Four **graph database normal forms** (1GNF, 2GNF, 3GNF, 4GNF).

b) These graph database normal forms are **organised into two levels of conceptual modelling**: data and metadata.

c) Moreover, the room for **yet one** conceivable graph normal form will be proposed.

Our future research will focus on the further empirical approvement of our claims and programming algorithms for transforming the conceptual model according to our rules in CASE modelling tools.

MOTIVATION: GQL – ISO/IEC 39075
2019 – 2024



MOTIVATION

April 18, 2024 – ISO/IEC 39075:2024 – GQL (Graph Query Language)

The professional community agrees that graph databases do not require such complex query programming for business intelligence applications, and their data model is more advantageous for machine learning large language models.

However, these advantageous properties of graph databases do not appear automatically. Therefore, software developers are confronted with the question, "*How to design the best database structure?*"

In the recent past, thanks to the enthusiasm for the new technology, it was spread that graph databases do not need many formal techniques of conceptual modelling and that only importing the data from the original relational tables and making edges between them into the graph database engine was enough to have it better.

OBJECT CLASS NORMALISATION AMBLER'S APPROACH

In 2003, **Scott W. Ambler** has proposed notable initial ideas regarding object-oriented normalisation – for the OOPL and OODB.

In 2018, **Shen-Hung Lo** (MIT) developed seven steps for object-oriented normalisation. This approach was based on both Ambler's class normalisation and relational database normalisation concepts.

FRISENDAL'S APPROACH TO THE GRAPH NORMALISATION

Thomas Frisendal's approach (Copenhagen University) to the Graph Normal Forms extends the classical relational normal forms without the distinction between the 3rd NF and Boyce-Codd NF and is expressed using the ISO/IEC 24707 standard: *Common Logic - A framework for a family of logic-based languages*.

- NF1: **Eliminate Repeating Groups.**
- NF2: **Eliminate Redundant Data.**
- NF3: **Eliminate Columns Not Dependent on Key.**
- NF4: **Isolate Independent Multiple Relationships.**
- GNF: **Remaining Functional Dependencies Rule.**

Frisendal explains that there must be recognised two levels of identities and uniqueness in graph databases (e.g. Neo4j and Memgraph): Business-level entities and physical-level keys with automatic IDs. For this reason, conceptual modelers must be sure that they have business-level identities on everything – no single attribute concept should be independent of business-level identities.

OUR APPROACH TO THE GRAPH NORMALISATION

Our approach is based on Ambler's three normal forms of object-oriented design, modified in the spirit of Frisendal's proposals.

We propose one more level of data types (metadata) above the basic level of data values. These two levels are arranged symmetrically, following the philosophy of Mendeleev's table of chemical elements, with the same rule being applied to data values as well as data types.

data types (metadata level)	**1st GNF** no repeating	**4th GNF** no sharing datatypes (inheritance)	**5th GNF** no independent datatypes (???)
data values (data level)		**2nd GNF** no sharing data	**3rd GNF** no independent data

EXAMPLE: UNSTRUCTURED DATABASE

Person
+name: String +surname: String +birthdate: Date +country +city +street +1st car label: String +1st car model: String +1st car range: Number +1st car emissions: Number +1st car made: Date +1st car producer name: String +1st car producer fullname: String +1st car producer country: String +1st car producer city: String +1st car producer street: String +2nd car label +2nd car model: String ...

This is the situation before normalisation as it can appear printed on paper when collecting software requirements from requesters who want such a database, for example.

In graph databases, there are two basic modelling concepts:

1) **nodes** (analogous to objects from OOPL) and
2) **edges** (analogous to object connections from OOPL

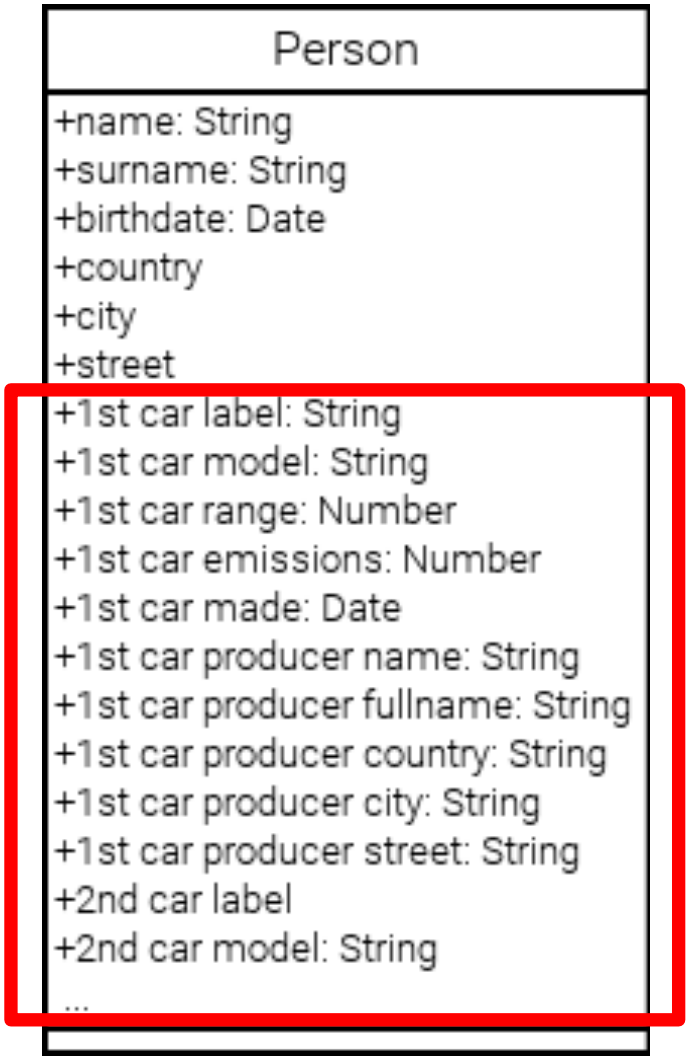
1ST GNF RULE – NO REPEATING DATA

There should be no repeating attributes within a single node.

Rule 1. *A node is in the first graph normal form (1st GNF) when it does not contain a group of repeating values. Groups of repeating values must be extracted into new nodes and linked to the original node by the edges. A database schema is in the 1st GNF when all nodes are in the 1st GNF.*

Definition 1. Let us have a node a, where for $k \geq 1$ (k is the length of the group of repeating data) and $n > 1$ (n is the number of repetitions of the group of repeating data) as $data(a) = \{\dots, x_1^1, \dots, x_1^k, \dots, x_n^1, \dots, x_n^k, \dots\}$, having $\forall i \in (1, \dots, k)$: $type(x_1^i) = type(x_2^i) = \dots = type(x_n^i)$. Then, it is required to extract these repeating data groups from the node a and store them in new nodes b_j for $j \in (1, \dots, n)$ as $data(b_j) = \{x_j^1, \dots, x_j^k\}$ and new edges $[a \rightarrow b_j]$.

CONFLICT WITH THE 1ST GNF

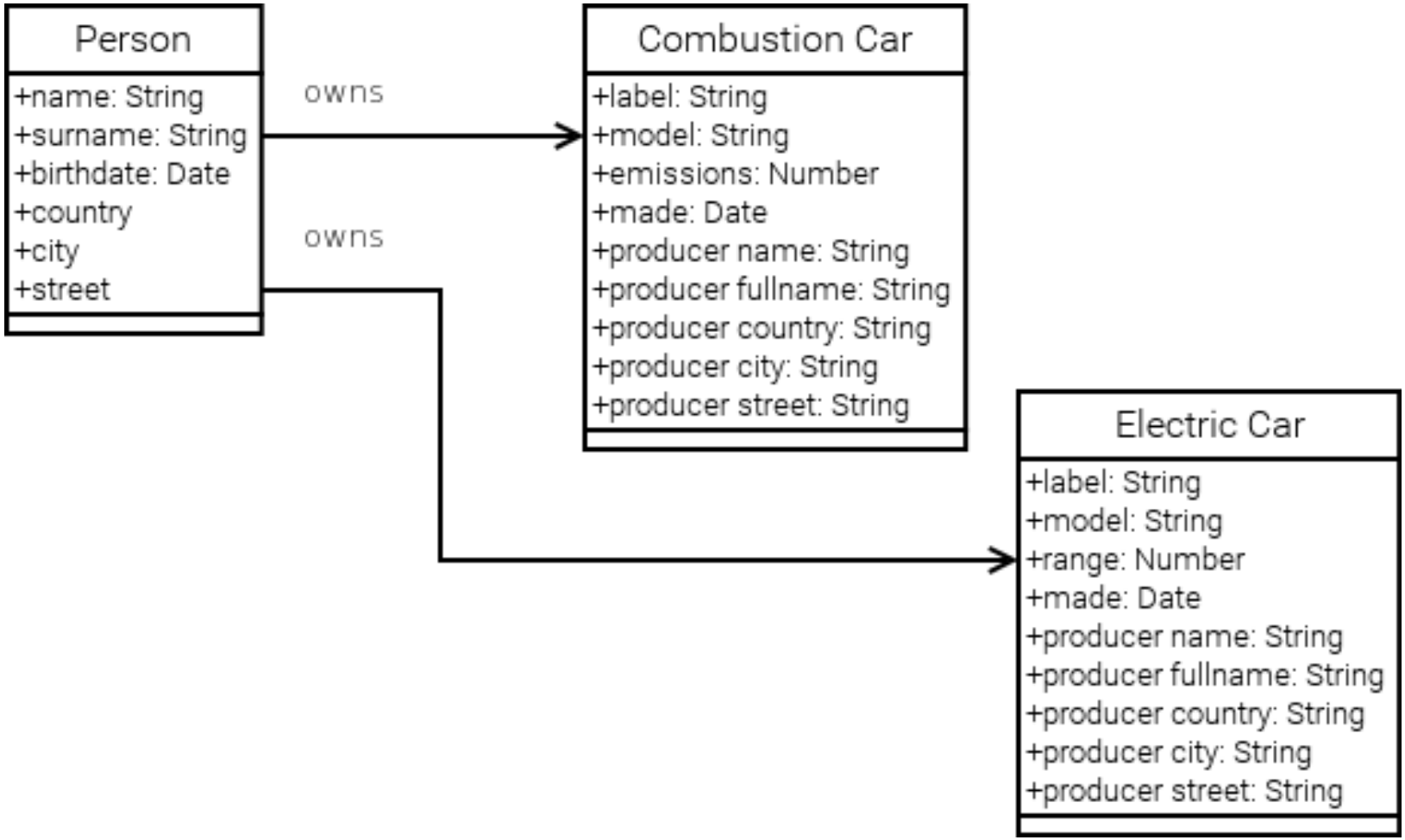


Repeating the conceptually same attributes within one node brings practical problems with data storage and subsequent querying.

For example:

1) We are limited by the maximum number of cars per person.
2) When searching for a person by car attributes, we must consider the order of cars of this person.

1ST GNF SOLUTION



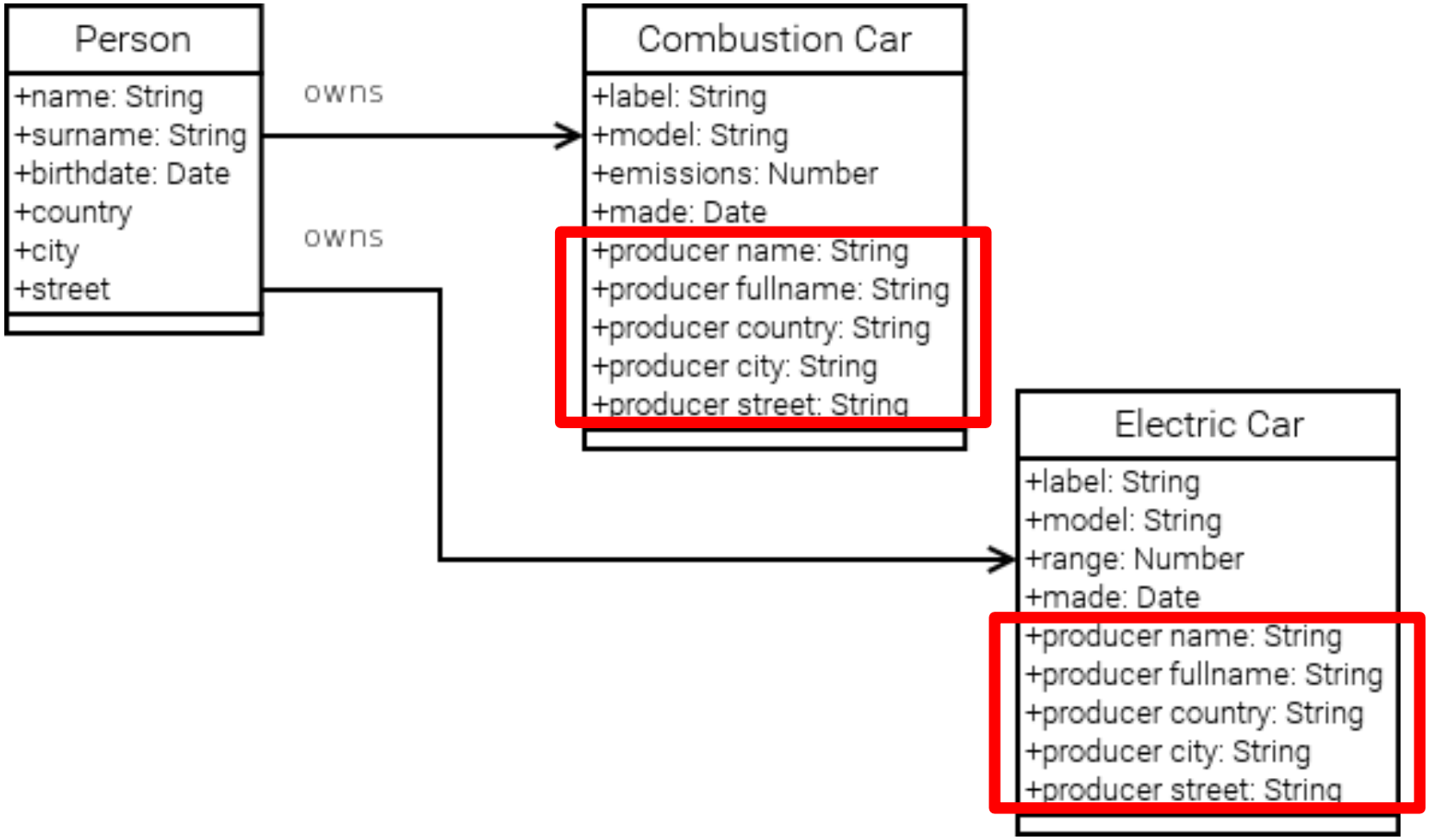
2ND GNF RULE – NO DATA SHARING

The identical data must be stored in the database only once and should only be accessible from all nodes where it is needed.

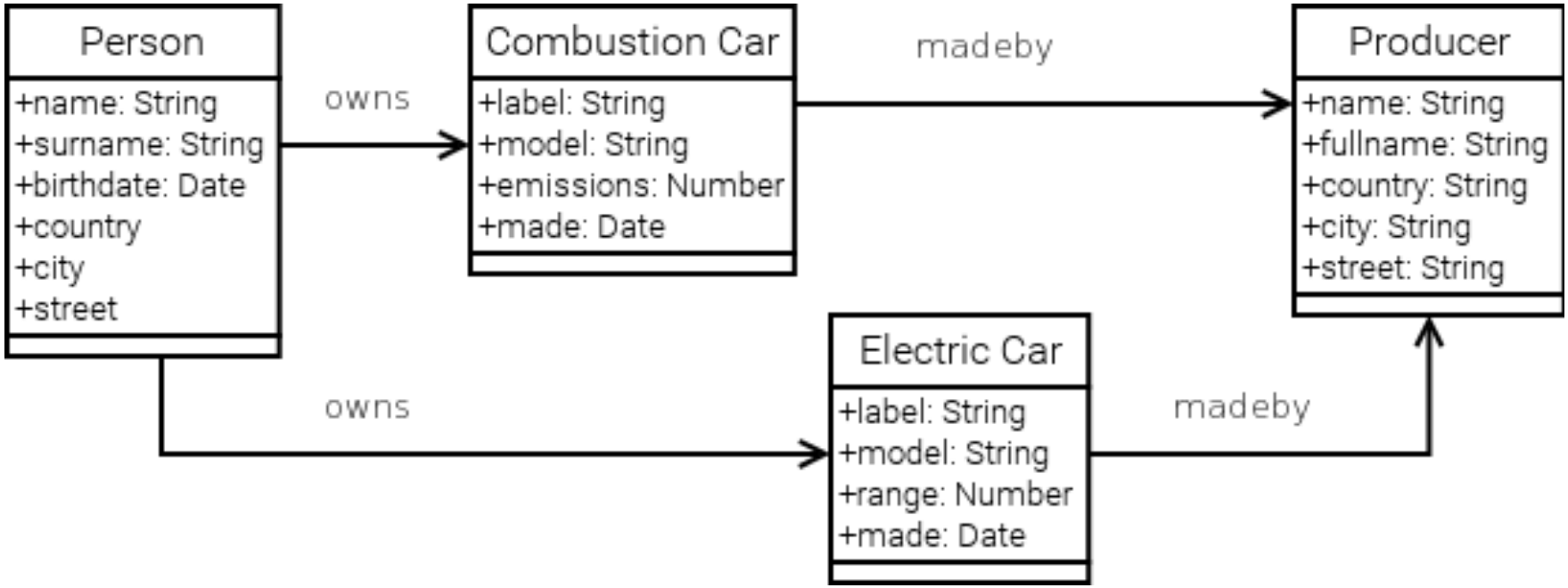
Rule 2. *Nodes are in the second graph normal form (2nd GNF) when they are in the 1st GNF and when they do not share identical data. Shared data must be extracted into new nodes and linked to the original nodes by the edges. A database schema is in the 2nd GNF when all nodes are in the 2nd GNF.*

Definition 2. Let us have two nodes a_1, a_2 for $k \geq 1$ (length of a group of shared data) as $data(a_1) = \{\dots, x_1, \dots, x_k, \dots\}$ and $data(a_2) = \{\dots, y_1, \dots, y_k, \dots\}$ having $\forall i \in (1, \dots, k)$: $x_i \equiv y_i$. Then it is required to extract these shared data from nodes a_1, a_2 to create new node b as $data(b) = \{x_1, \dots, x_k\}$ and create new edges $[a_1 \rightarrow b]$, $[a_2 \rightarrow b]$.

CONFLICT WITH THE 2ND GNF



2ND GNF SOLUTION



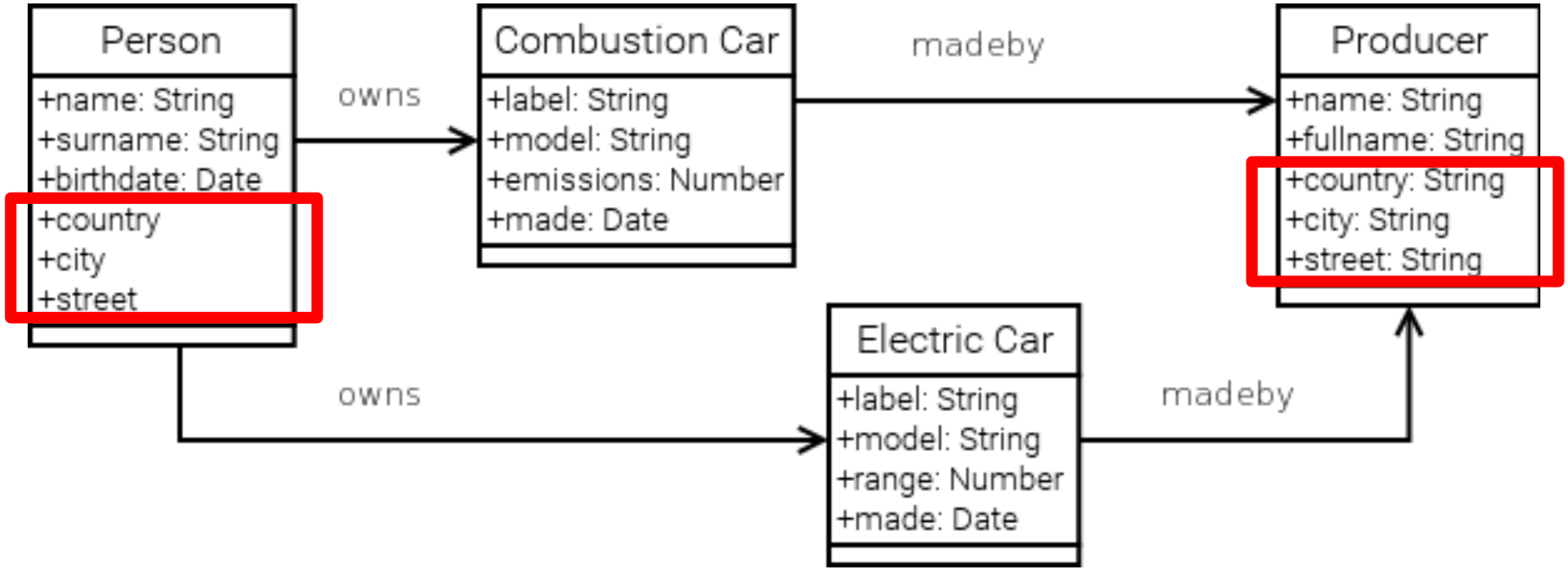
3RD GNF RULE – NO INDEPENDENT DATA

No **data** should be stored in nodes on which it does not depend.

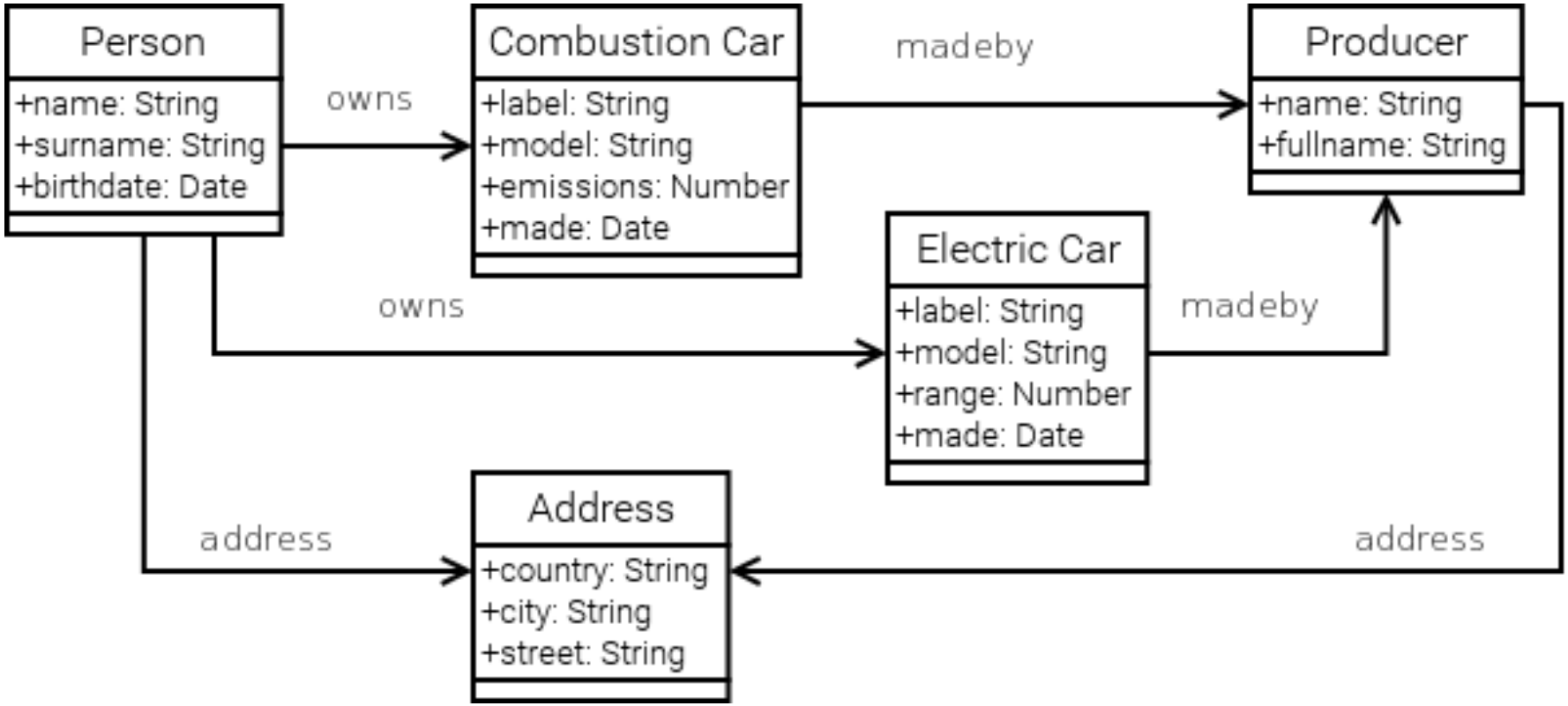
Rule 3. *A node is in the third graph normal form (3rd GNF) when it is in the 2nd GNF and when it does not contain a value or a group of values, which have an independent interpretation of the node identity. The independent values must be extracted into a new node and linked to the original node by an edge. A database schema is in the 3rd GNF when all nodes are in the 3rd GNF.*

Definition 3. Let us have a node a for $k \geq 1$ (length of a group of independent data) having $data(a) = \{\dots, x_1, \dots, x_k, \dots\}$, where $\{x_1, \dots, x_k\}$ is a group of independent data. Then, it is required to extract this group of independent data from the node a, and create a new node b as $data(b) = \{x_1, \dots, x_k\}$ and new edge $[a \rightarrow b]$.

CONFLICT WITH THE 3RD GNF



3RD GNF SOLUTION



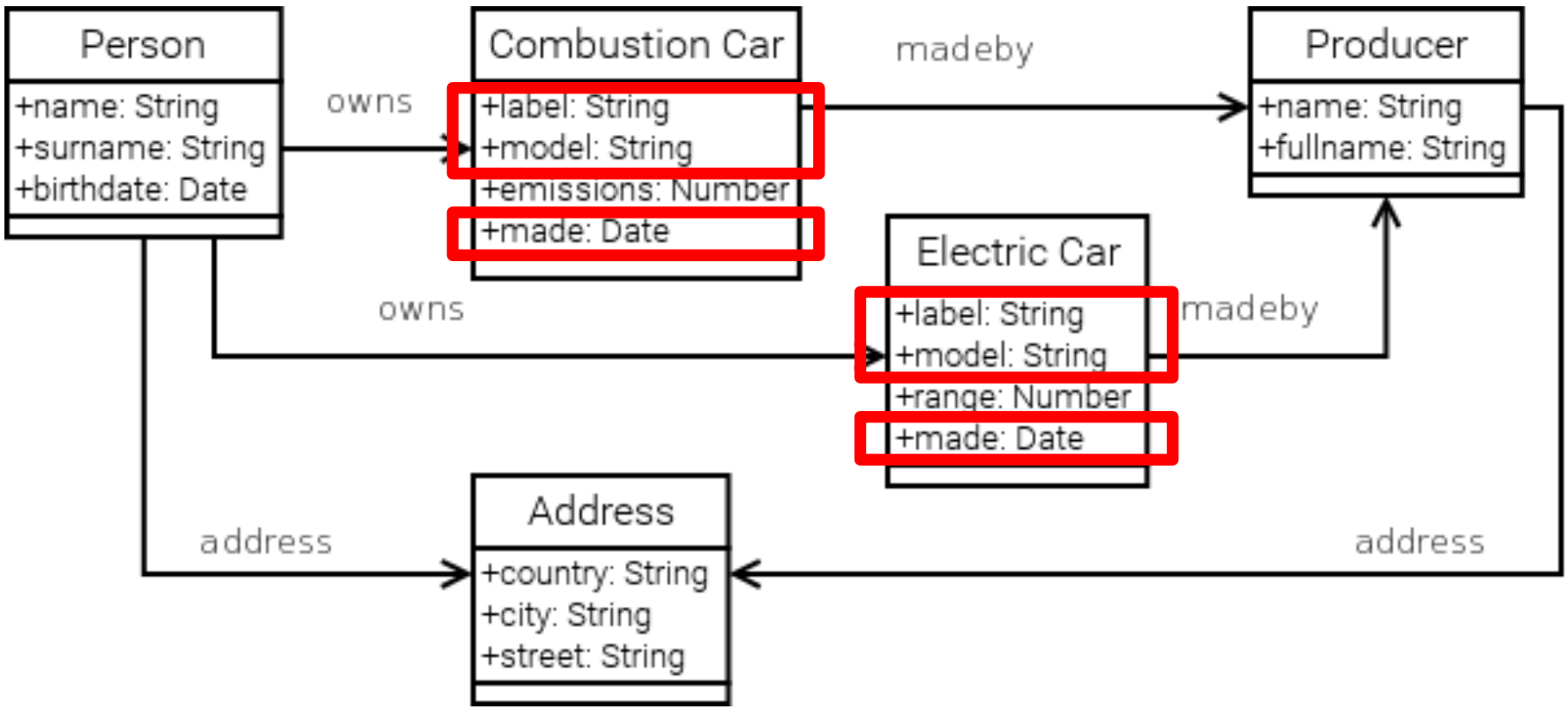
4TH GNF RULE – NO DATA TYPES SHARING

The identical data types must be stored in the database only once and should only be accessible from all nodes where it is needed.

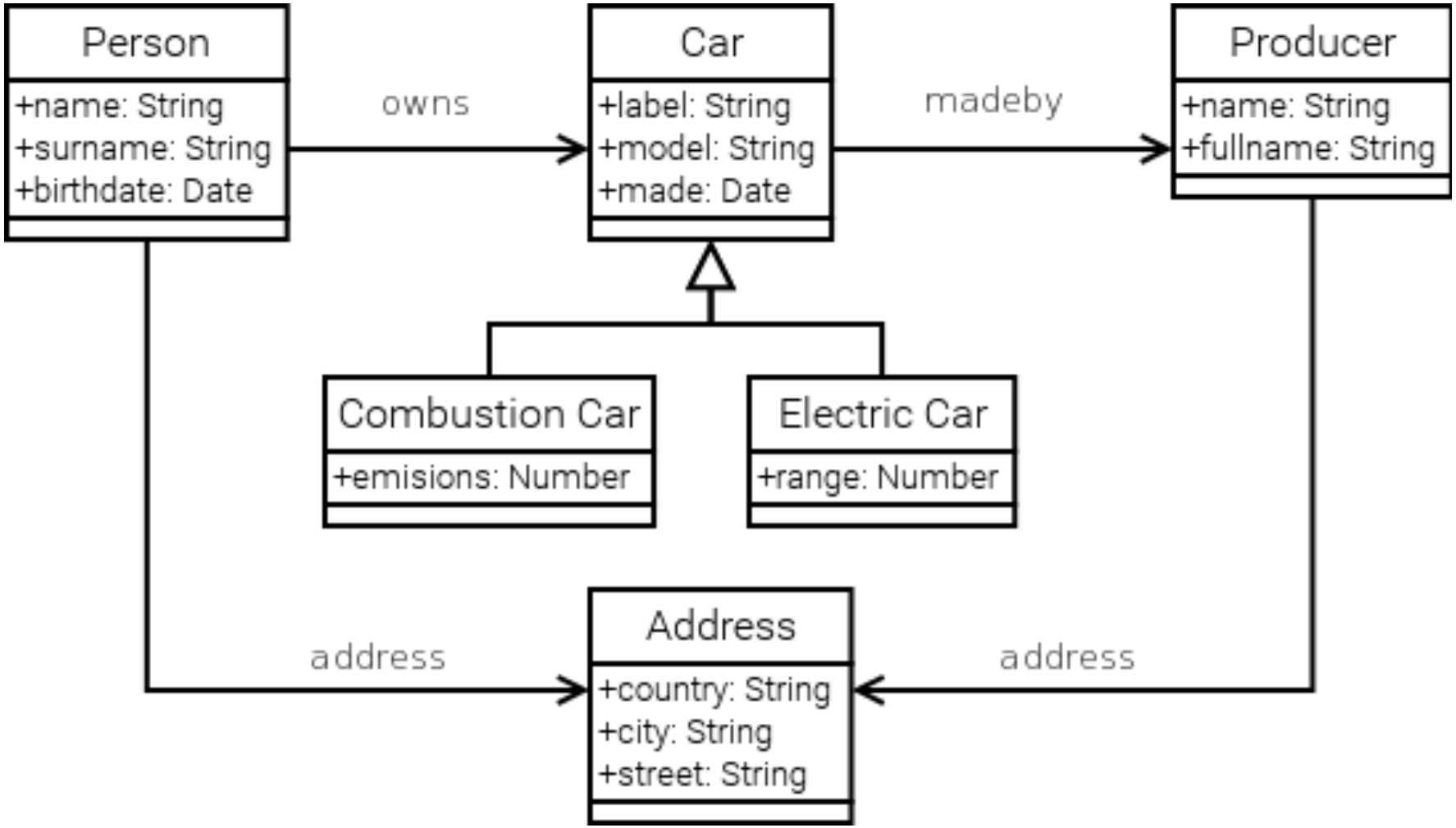
Rule 4. *Nodes are in the fourth graph normal form (4th GNF) when they are in the 3rd GNF and when their node types do not share identical data types. Shared data types must be extracted into a new node type and linked to the original node type by inheritance. A database schema is in the 4th GNF when all nodes are in the 4th GNF. (The term node type is analogous to the term object class in object-oriented programming.)*

Definition 4. Let us have two node types a_1, a_2 for $k \geq 1$ (length of a group of shared types) as $datatypes(a_1) = \{\dots, type(x_1), \dots, type(x_k), \dots\}$ and $datatypes(a_2) = \{\dots, type(y_1), \dots, type(y_k), \dots\}$ having $\forall i \in (1, \dots, k)$: $type(x_i) \equiv type(y_i)$. Then it is required to extract these shared data types from node types a_1, a_2 to create a new node type b as $datatypes(b) = \{type(x_1), \dots, type(x_k)\}$ and create new inheritance links between node types a_1, a_2, b as $[a_1 \succ b]$ and $[a_2 \succ b]$.

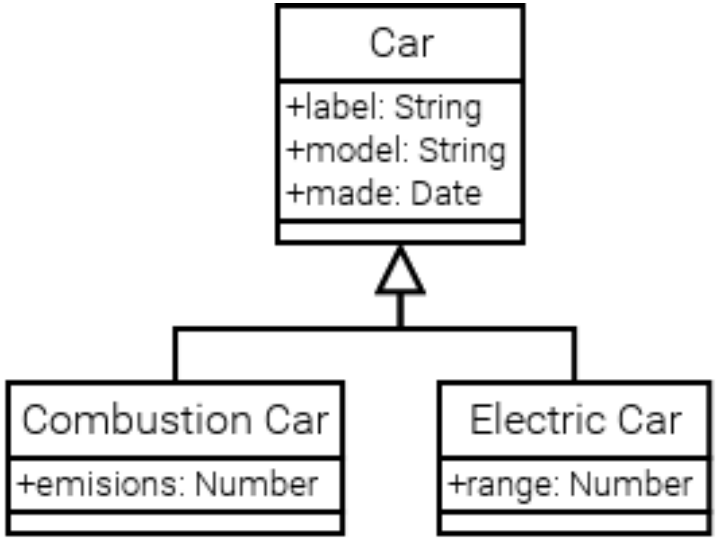
CONFLICT WITH THE 4TH GNF



4TH GNF SOLUTION



IS THERE THE INHERITANCE IN GDB?



GQL example of data creation:

```
CREATE (car1:Car:Combustion
        {label: "9F7 9987" , model: "Kuga ST-Line"})
CREATE (car2:Car:Electric
        {label: "CA 673-998", model: "Tesla S"})
```

IS THERE THE INHERITANCE IN GDB?

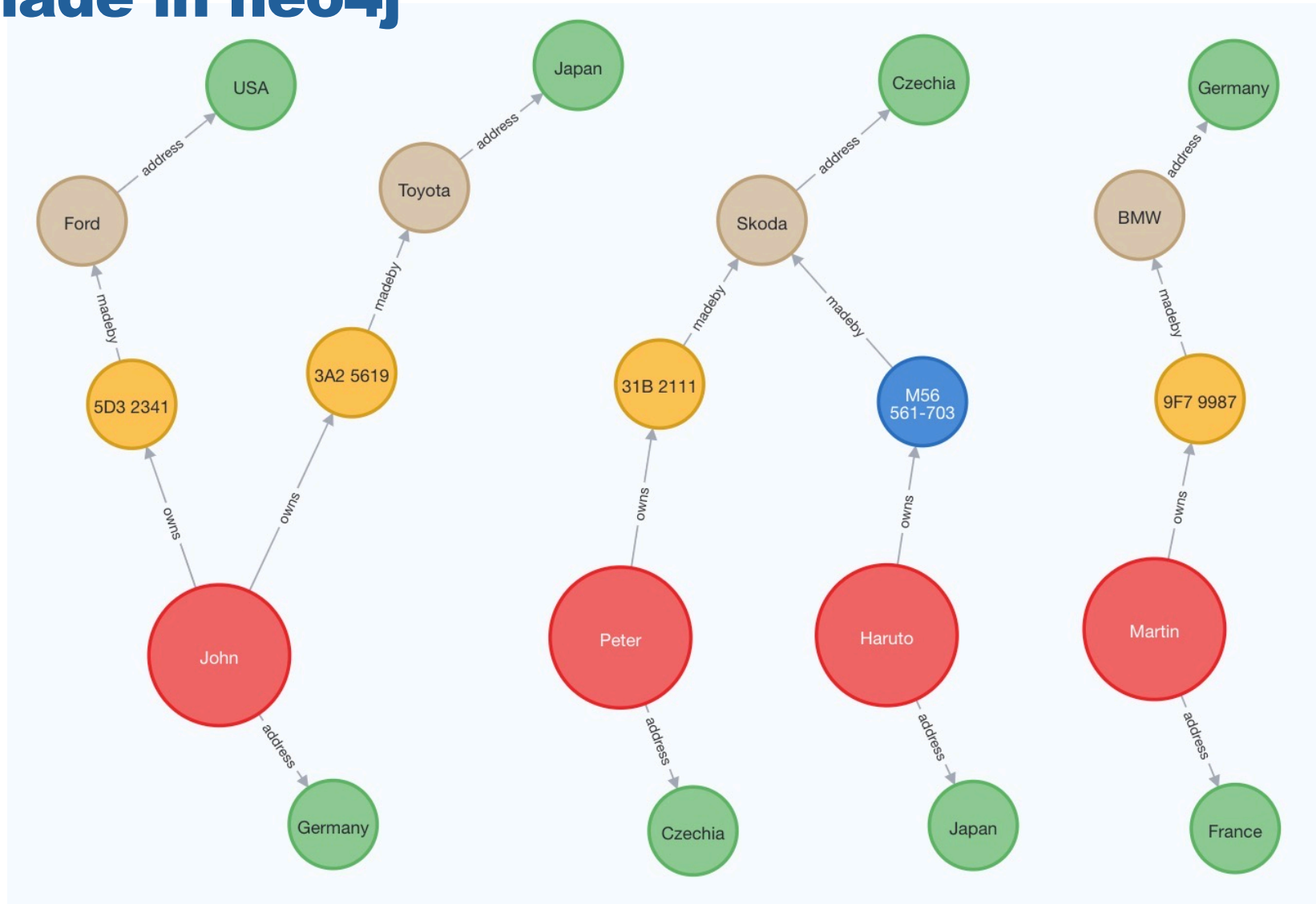
GQL example of a selection of only electric cars belonging to any owner:

```
MATCH
 (addr:Address)<-[:address]-(owner:Person)-[:owns]->(car:Electric)
RETURN
 owner.name , owner.surname , addr.city , addr.country , car.model
```

GQL example of a selection of all kinds of cars with their producers:

```
MATCH
  (car:Car)-[:madeby]->(prod:Producer)-[:address]->(addr:Address)
RETURN
  car.label , car.model , prod.name , addr.country
```

GDB EXAMPLE
made in neo4j



BENEFITS OF NORMALISED GDB 1/3

GQL example of searching for patriots
(i.e. owners of any kind of car manufactured in the same country where they live)

```
MATCH
  (addr1:Address)<-[:address]-(owner:Person)-[:owns]->
  (car:Car)-[prod:Producer]->(addr2:Address)
WHERE
  addr1.country = addr2.country
RETURN
  owner.name , owner.surname , addr1.country , car.model , prod.name
```

This query would require much greater programming effort in RDB,
going as far as software add-ons of the business intelligence type.

BENEFITS OF NORMALISED GDB 3/2

GQL example of searching for people who have something in common with Japan:

```
MATCH path = ((p:Person)-[*]-(addr:Address))
WITH p , nodes(path) AS nodes
WHERE addr.country = "Japan"
      AND size([node IN nodes WHERE ("Car" IN labels(node))]) <= 1
RETURN
(p.name + ' ' + p.surname) as person , nodes as connection_to_Japan
```

We did not have to know the path of edges through internal nodes; instead, we just asked for that unknown path and obtained various kinds of path results:

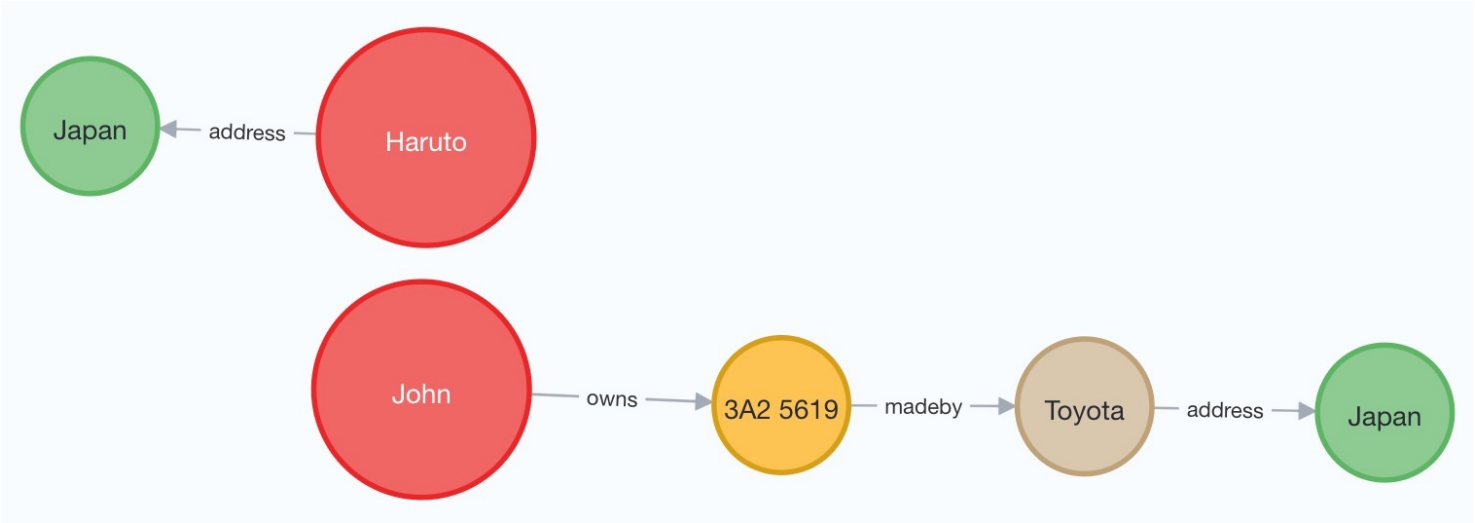
a) **Persons living in Japan** will be found.

b) Another path will also be found, as **persons** living anywhere but **owning a Japanese car**.

We restricted the properties of internal path nodes to at most one car of any car type.

BENEFITS OF NORMALISED GDB 3/3

person	connection_to_Japan
"Haruto Tanaka"	[(:Person {surname: "Tanaka", name: "Haruto"}) , (:Address {country: "Japan", city: "Tokyo"})]
"John Smith"	[(:Person {surname: "Smith", name: "John"}) , (:Car {model: "RAV4 GR", label: "3A2 5619"}) , (:Producer {name: "Toyota"}) , (:Address {country: "Japan", city: "Toyota City"})]



CONCLUSION

Based on our experience, our design allows development team members to improve the quality of their software engineering work, reduce uncertainty, and improve conceptual consistency to better support graph database properties.

1) We added inheritance to the graph normalisation rules, which is currently not supported in graph databases but is present in most programming languages on the client side that access graph databases.
2) We have shown new conceptual connections (analogy, metamodelling symmetry) between **data composition** and **data inheritance**.
3) We may have found room for a **new inheritance-like concept** of modelling (analogous to the 3rd GNF in the data level as the 5th GNF in the metadata level).

It remains to decide what can be hidden under the expected 5th GNF. Incomplete classes called **mixins** are offered as possible candidates. In the conceptual modelling techniques, it corresponds to the **decorator** (or **wrapper**) design pattern style, where there are added more properties to existing object classes. However, since today's graph databases do not directly support the inheritance of object classes, we leave this issue open for the future.

Thank you for your attention.