

# Data And Application

**Project Phase - 3**

**Team No. - 57**

**Team Name - Demolishers**

**Team members Details**

<b>Name</b>	<b>Roll no.</b>	<b>Email ID</b>
Abhinav Anand	2020101054	<a href="mailto:abhinav.a@students.iiit.ac.in">abhinav.a@students.iiit.ac.in</a>
Lavisha Bhambri	2020101088	<a href="mailto:lavisha.bhambri@students.iiit.ac.in">lavisha.bhambri@students.iiit.ac.in</a>
Mayank Bhardwaj	2020101068	<a href="mailto:mayank.bhardwaj@students.iiit.ac.in">mayank.bhardwaj@students.iiit.ac.in</a>

# Contents

## 1. Conversion from ER Model to Relational Model

- 1.1 Mapping Strong entity types to Relations
- 1.2 Mapping of Weak Entity types
- 1.3 Mapping of Binary 1:1 Relationship Types
- 1.4 Mapping of Binary 1:N Relationship types
- 1.5 Mapping of Binary M:N Relationship types
- 1.6 Mapping of Multi-valued Attributes
- 1.7 Mapping of N-ary Relationship Types
- 1.8 Mapping of Superclasses and Subclasses

## 2. Normal Forms

- 2.1 1<sup>st</sup> Normal Form
- 2.2 2<sup>nd</sup> Normal Form
- 2.3 3<sup>rd</sup> Normal Form

# **Task 1: Conversion from ER Model to Relational Model**

## **1.1 Mapping Strong entity types to Relations**

For every regular strong entity type **E** in the schema, we created a relation that includes all simple attributes of **E**.

Employee		Insurance Policy		Third Party Administrator		Customer	
PK	<u>department no</u>	PK	<u>policy id</u>	PK	<u>TPA id</u>	PK	<u>aadhar no</u>
PK	<u>serial no</u>		customer_name		TPA_name		date_of_birth
	<u>aadhar_no</u>		terms_and_conditions		street_address		first_name
	date_of_birth		date_of_issue		zip_code		middle_name
	first_name		duration		city		surname
	middle_name		Premium Value		state		email_id
	surname		Sum assured				customer_status
	email_id						street_address
	street_address						zip_code
	zip_code						city
	city						state
	state						age
	age						

Figure 1.1: Mapping Strong Entites as Relations

## 1.2 Mapping of Weak Entity types

For every weak entity type **W** in the ER schema with owner entity type **E**, we have created a relation and included all simple attributes as attributes of relations. We also included the primary key of the owner entity type **E** as the foreign key of the relation. The primary key of **W**'s relation is the combination of **E**'s primary key and partial key of **W**.

```

    erDiagram
        Employee-Dependant ||--o{ Employee : "has"
        Employee-Dependant ||--o{ Insurance-Policy : "has"
        Employee-Dependant ||--o{ Customer : "has"
        Employee ||--o{ Insurance-Policy : "has"
        Employee ||--o{ Customer : "has"
        Insurance-Policy ||--o{ Customer : "has"

        Employee-Dependant {
            string first_name PK
            string middle_name PK
            string surname PK
            string employee_department_no FK
            string employee_serial_no FK
            string date_of_birth
            int age
        }
        Employee {
            string department_no PK
            string serial_no PK
            string aadhar_no
            string date_of_birth
            string first_name
            string middle_name
            string surname
            string email_id
            string street_address
            string zip_code
            string city
            string state
            int age
        }
        Insurance-Policy {
            string policy_id PK
            string customer_name
            string terms_and_conditions
            string date_of_issue
            int duration
            float Premium_Value
            float Sum_assured
        }
        Customer {
            string aadhar_no PK
            string date_of_birth
            string first_name
            string middle_name
            string surname
            string email_id
            string customer_status
            string street_address
            string zip_code
            string city
            string state
            int age
        }
  
```

The diagram illustrates the following tables and their attributes:

- Employee Dependant** (Red border):
  - PK: first\_name, middle\_name, surname
  - FK: employee\_department\_no, employee\_serial\_no
  - date\_of\_birth
  - age
- Employee** (Red border):
  - PK: department\_no, serial\_no
  - aadhar\_no
  - date\_of\_birth
  - first\_name
  - middle\_name
  - surname
  - email\_id
  - street\_address
  - zip\_code
  - city
  - state
  - age
- Insurance Policy** (Blue border):
  - PK: policy\_id
  - customer\_name
  - terms\_and\_conditions
  - date\_of\_issue
  - duration
  - Premium Value
  - Sum assured
- Third Party Administrator** (Green border):
  - PK: TPA\_id
  - TPA\_name
  - street\_address
  - zip\_code
  - city
  - state
- Customer** (Blue border):
  - PK: aadhar\_no
  - date\_of\_birth
  - first\_name
  - middle\_name
  - surname
  - email\_id
  - customer\_status
  - street\_address
  - zip\_code
  - city
  - state
  - age
- Customer Dependant** (Blue border):
  - PK: first\_name, middle\_name, surname
  - FK: customer\_aadhar\_no
  - date\_of\_birth
  - age

Figure 1.2: Mapping Weak Entities as Relations

### 1.3 Mapping of Binary 1:1 Relationship Types

Since we don't have any binary 1:1 relationships, we do not need to do anything.

<b>Employee Dependant</b>	
PK	<u>first_name</u>
PK	<u>middle_name</u>
PK	<u>surname</u>
FK	<u>employee_department_no</u>
FK	<u>employee_serial_no</u>
	date_of_birth age

<b>Employee</b>	
PK	<u>department_no</u>
PK	<u>serial_no</u>
	aadhar_no  date_of_birth  first_name  middle_name  surname  email_id  street_address  zip_code  city  state  age

<b>Insurance Policy</b>	
PK	<u>policy_id</u>
	customer_name terms_and_conditions date_of_issue duration Premium Value Sum assured

<b>Third Party Administrator</b>	
PK	<u>TPA_id</u>
	TPA_name street_address zip_code city state

<b>Customer</b>	
PK	<u>aadhar_no</u>
	date_of_birth first_name middle_name surname email_id customer_status street_address zip_code city state age

<b>Customer Dependant</b>	
PK	<u>first_name</u>
PK	<u>middle_name</u>
PK	<u>surname</u>
FK	<u>customer_aadhar_no</u>
	date_of_birth age

Figure 1.3: Mapping all Binary 1:1 Relationship types

## 1.4 Mapping of Binary 1:N Relationship types

For each binary 1:N relationship, in the entity with cardinality 1 we add the primary key of the other entity as a foreign key and map these keys to each other.

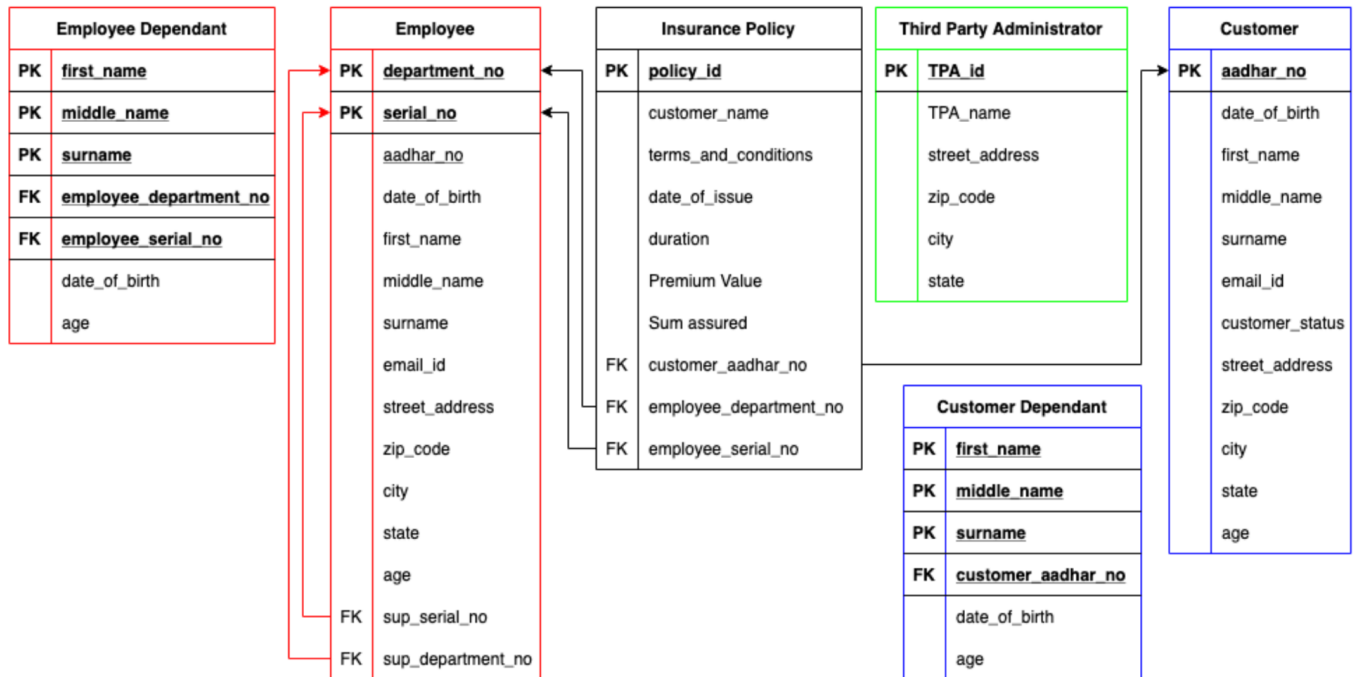


Figure 1.4: Mapping all Binary 1:N Relationship types

## 1.5 Mapping of Binary M:N Relationship types

For each binary M:N relationship type **R**, we are creating a new relationship relation **S** which includes the primary keys of participating entities as foreign keys.

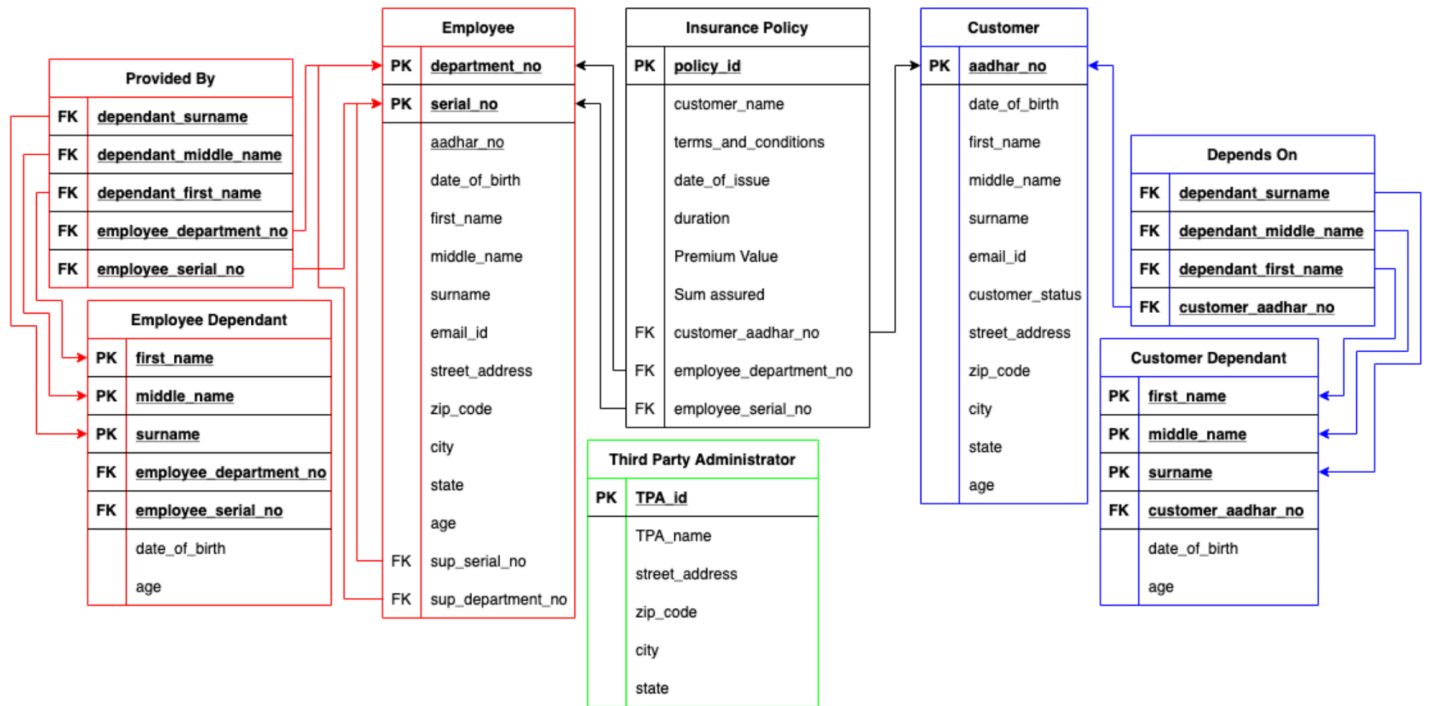


Figure 1.5: Mapping all Binary M:N Relationship types

## 1.6 Mapping of Multi-valued Attributes

For each multi-valued attribute **A**, create a new relation **R** that will include attributes corresponding to **A**, the primary key attribute **K** as the foreign key in **R**. The primary key is a combination of **A** and **R**.

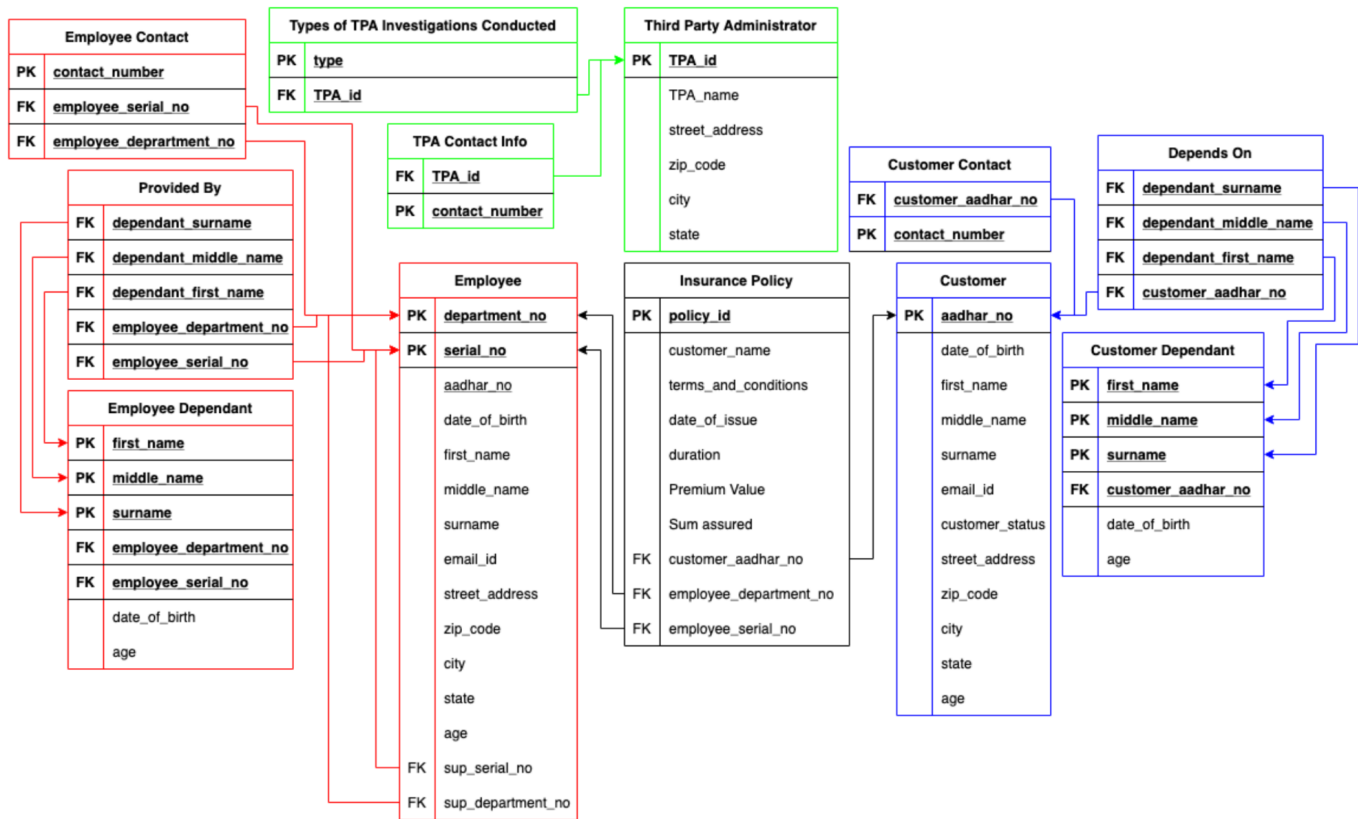


Figure 1.6: Mapping all multivalued attributes



## 1.7 Mapping of N-ary Relationship Types

For each n-ary relationships, we create a relation **R** which has the primary keys of the participating entities as its foreign keys, and primary key. **R** also contains the attributes of the n-ary relationship.

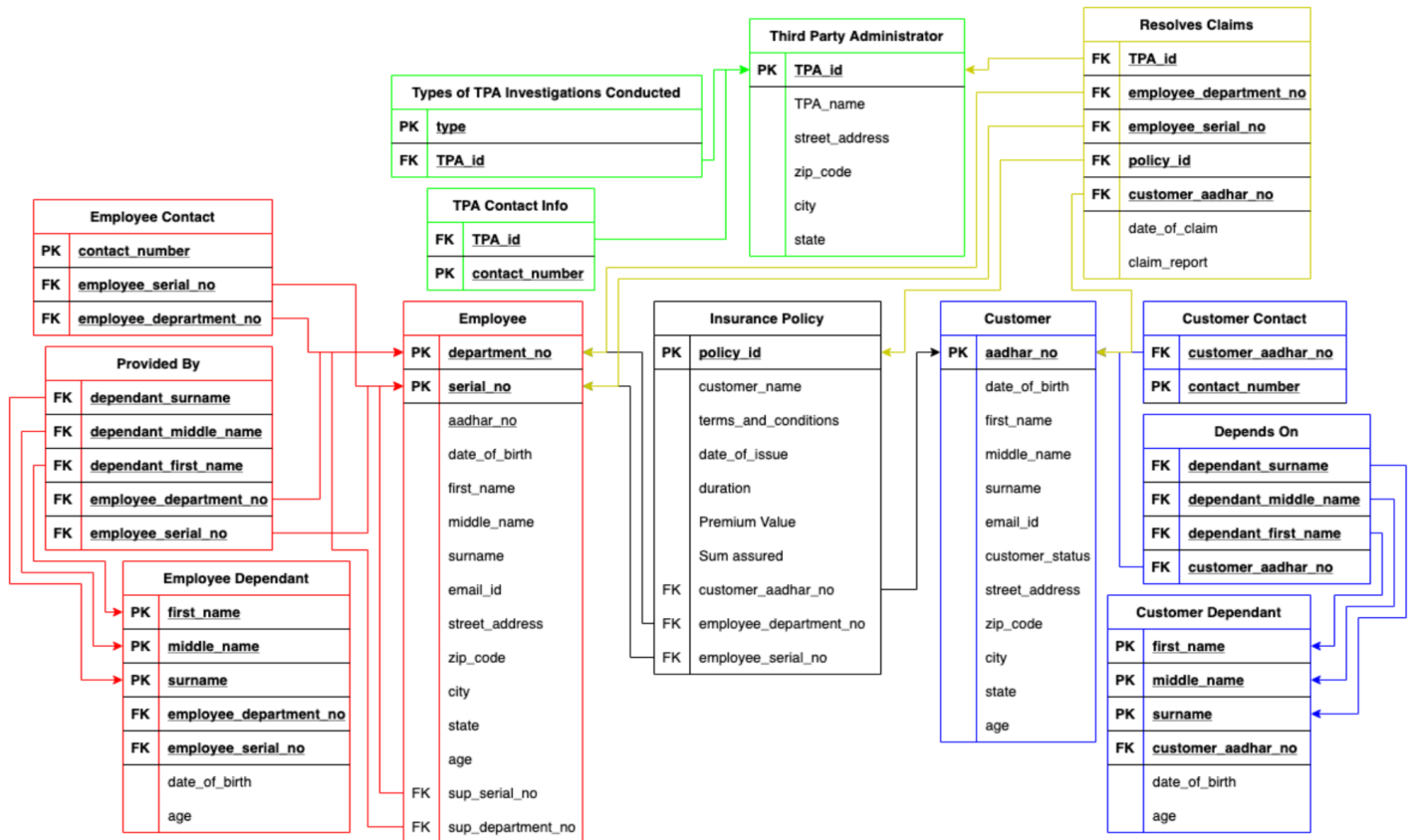


Figure 1.7: Mapping all N-ary Relationship types

## 1.8 Mapping of Superclasses and Subclasses

For every subclass of the superclass **Insurance Policy** we create a new relation whose primary keys and foreign keys are the primary key of its superclass **Insurance Policy**. We add their simple attributes in these relations and then we follow steps 2-7 for these newly created relations.

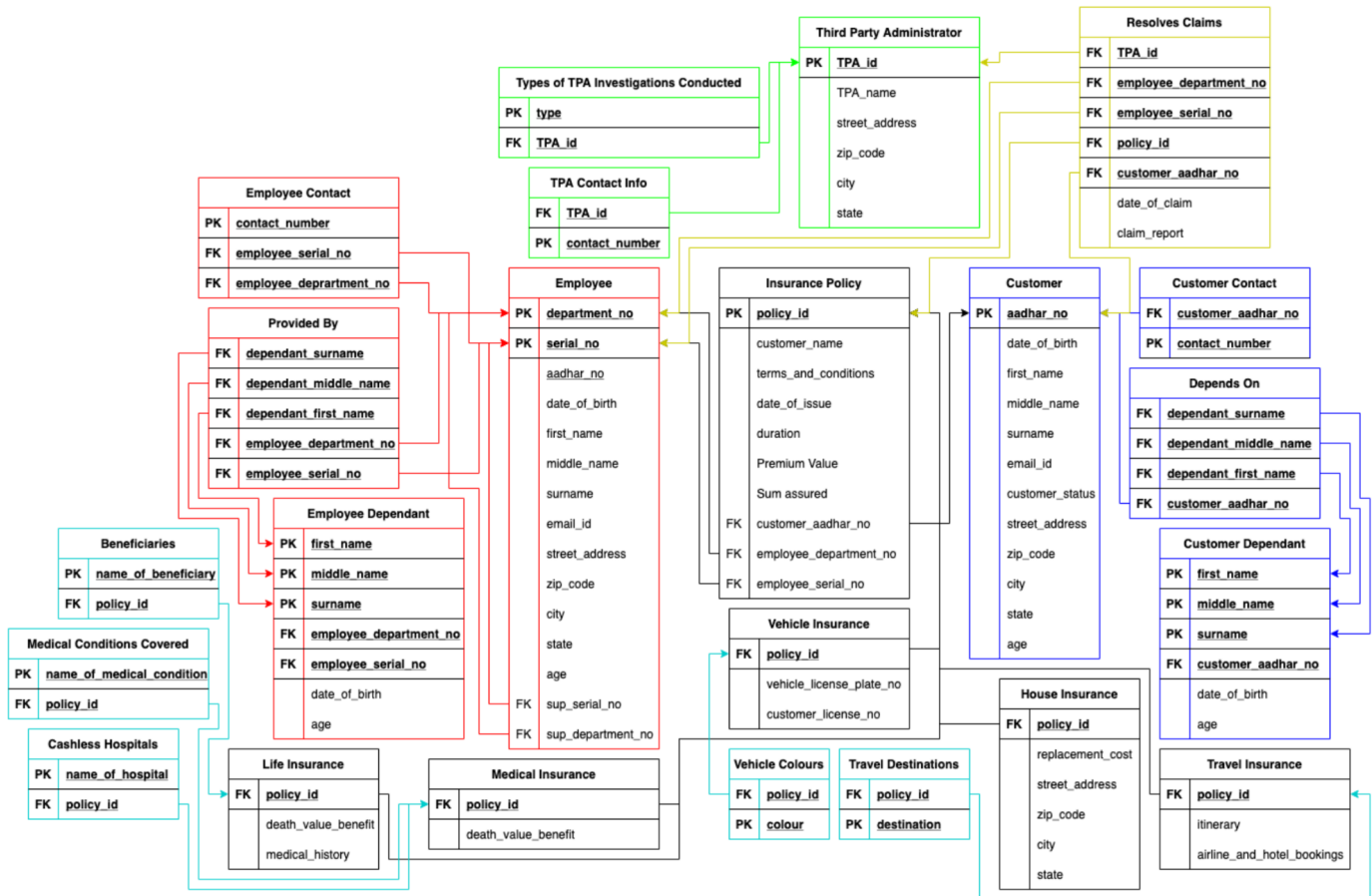


Figure 1.8: Mapping all Superclasses and Subclasses

## **Task 2: Normal Forms**

### **2.1 1<sup>st</sup> Normal Form**

We have handled all cases of multivalued and composite attributes, and we don't have any nested relations in our model. Hence, our model already is in 1 NF form.

## 2.2 2<sup>nd</sup> Normal Form

All the relations that have a single attribute in the primary key do not have an attribute to remove. Hence, these relations are fully functional dependent and can be ignored while checking for partially functional dependent relations.

The only relation that fails the 2 Normal Form test is **Resolves Claims**.

This is because the set of attributes (**claim\_report**, **date\_of\_claim**) can be functionally determined by the attribute **policy\_id** of the primary key.

This relation can be secondly normalized by performing the following steps:

- Delete the attributes **date\_of\_claim** and **claim\_report** from the relation **Resolves Claims**.
- Create a new Relation **Resolves\_Claims\_Attributes**(**policy\_id**, **date\_of\_claim**, **claim\_report**) where the attribute **policy\_id** is the primary key and the foreign key.

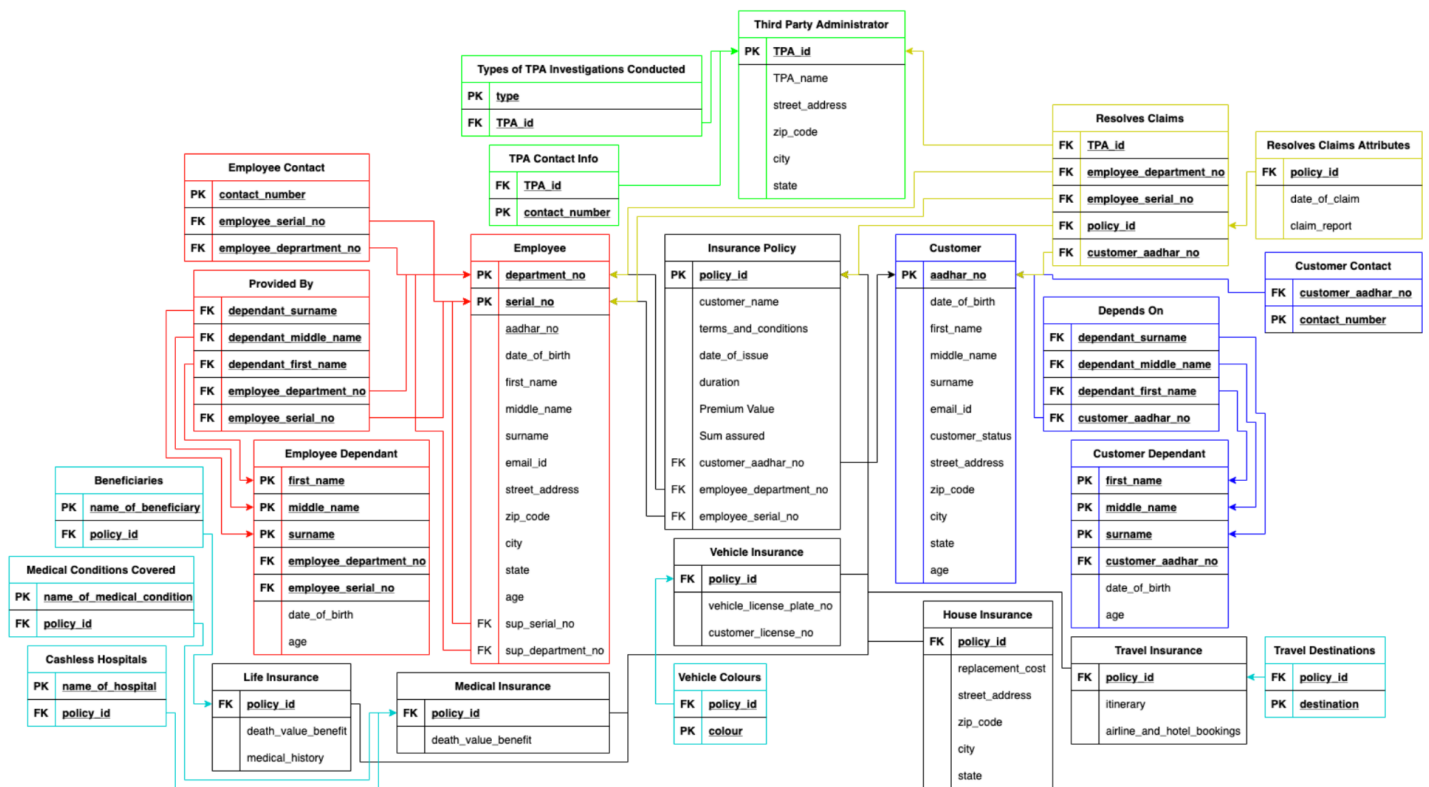


Figure 2.1: Mapping After Second Normalization

## 2.3 3<sup>rd</sup> Normal Form

In the relations **Employee**, **Employee Dependent**, **Customer** and **Customer Dependent**, the attribute **age** is functionally dependent on the attribute **date\_of\_birth**, which is in turn functionally dependent on the primary key.

In the relation **Resolves Claims Attributes**, the attribute **claim\_report** is functionally dependent on the attribute **date\_of\_claim**, which is in turn functionally dependent on the primary key.

In relation **Vehicle Insurance**, the attribute **customer\_license\_info** is functionally dependent on the attribute **vehicle\_license\_plate\_no**, which is in turn functionally dependent on the primary key.

This can be solved by the following steps:

- Delete the attribute **age** from the relation **Employee** and create a new relation **Employee\_Age**(age, employee\_serial\_no, employee\_department\_no) where all the attributes are the primary key, and the attributes **employee\_serial\_no** and **employee\_department\_no** are foreign keys.
- Delete the attribute **age** from the relation **Customer** and create a new relation **Customer\_Age**(age, customer\_aadhar\_no) where all the attributes are the primary key, and the attribute **customer\_aadhar\_no** is the foreign key.
- Delete the attribute **age** from the relation **Employee Dependent** and create a new relation **Employee\_Dependent\_Age**(age, dependent\_first\_name, dependent\_middle\_name, dependent\_surname) where all the attributes are the primary key, and the attributes **dependent\_first\_name**, **dependent\_middle\_name** and **dependent\_surname** are foreign keys.
- Delete the attribute **age** from the relation **Customer Dependent** and create a new relation **Customer\_Dependent\_Age**(age, dependent\_first\_name, dependent\_middle\_name, dependent\_surname) where all the attributes are the primary key, and the attributes **dependent\_first\_name**, **dependent\_middle\_name** and **dependent\_surname** are foreign keys.
- Rename the relation **Resolves Claims Attributes** to **Claim Date**, delete the attribute **claim\_report** from this relation and create a new relation **Claim\_Report**(policy\_id, claim\_report) where all the attributes are the primary key, and the attribute **policy\_id** is the foreign key.
- Delete the attribute **customer\_license\_no** from the relation **Vehicle Insurance** and create a new relation **Customer\_License\_No**(policy\_id, customer\_license\_no) where all the attributes are the primary key, and the attribute **customer\_license\_no** is the foreign key.