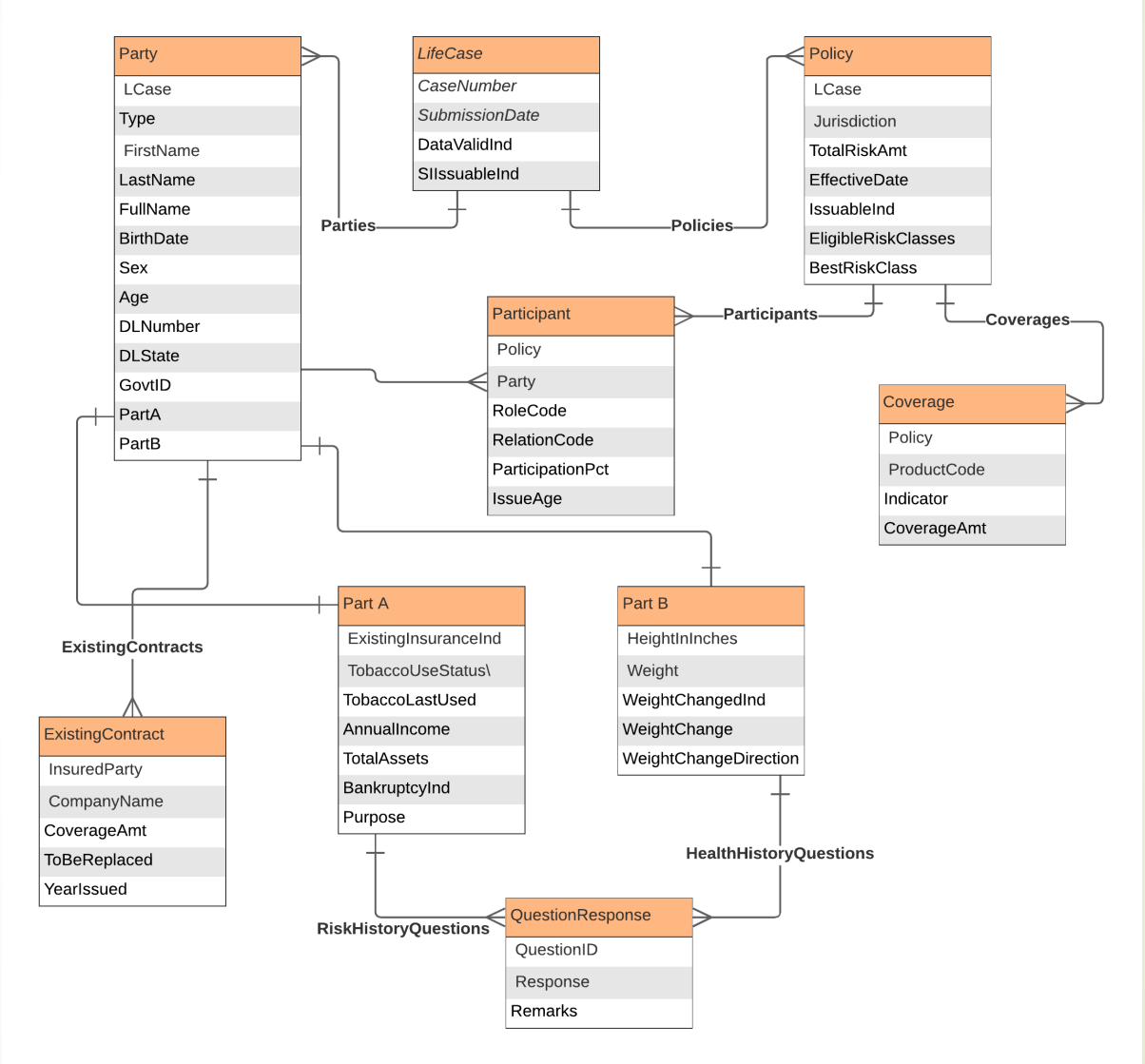
Data Validation Use Case

Introduction

- This use case illustrates how Maximal can be used to track required fields and other data validations when dependencies are complex. We use the data model from the Life Insurance Simplified Issue use case.
- The following are illustrated:
  - What are all the required fields in a life insurance application?
    - This logic can get complicated as the data requirements may vary vastly based on the product, state, and individual answers the applicant has given.
  - If an application is invalid, provide explanations as to why.
    - Think of this as something similar to a tax filing software providing you explanations for why it is asking certain questions.



Data Validation Logic

Some sample constraints to check:

- All required data fields must be specified.
- At least one beneficiary must be specified.
- The sum of all beneficiary participations must be 100%.
- If contingent beneficiaries are named, the sum of their participation must be 100%.
- If applicant has any existence insurance contracts, their details must be provided.

Some sample data items that must be provided;

- Insured Party: First Name, Last Name, Birth Date.
- If an insured party has valid driver license, DL Number and DL State must be provided.
- If an insured party stopped smoking, last tobacco use date and remarks must be provided.
- All risk history questions, and health history must be answered.
- If any of risk history and health history questions is answered yes, corresponding remarks must be provided.

Data Validation Logic

```
GROUP case.DataValidInd

  CONSTRAINT DataValidity C1 "No beneficiaries specified."
    SSIZE(bene)>0;

  CONSTRAINT DataValidity C2 "Beneficiary percentages should add up to 100."
    SSIZE(bene)>0 => SSUM(bene, bene.ParticipationPct) = 100;

  CONSTRAINT DataValidity C3 "Contingent Bene percentages should add up to 100."
    SSIZE(contbene)>0 => SSUM(contbene, contbene.ParticipationPct) = 100;

  CONSTRAINT DataValidity C4 "Existing insurance indicated, but no existing contracts
specified."
    partA.ExistingInsuranceInd => SSIZE(existingContract)>0;

/GROUP;
```

Set Sum operator used to sum up participation percentages for beneficiaries and contingent beneficiaries. When a beneficiary is added or deleted, the sets automatically maintain memberships and apply these constraints.

- It is quite easy to implement very complex data validations and missing field checks in Maximal.
  - As a result, Maximal is ideally suited building smart forms / e-apps, etc.

Data Completeness Logic

```
GROUP case.DataCompleteInd

  CONSTRAINT DataComplete C1 "All Insured Party data must be provided."
    BOUNDED(insuredParty.FirstName, insuredParty.LastName, insuredParty.BirthDate);

  CONSTRAINT DataComplete C2 "Driving License information must be provided."
    insuredParty.HasValidDL => BOUNDED(insuredParty.DLNumber, insuredParty.DLState);

  CONSTRAINT DataComplete C3 "Tobacco past usage data must be provided."
    (partA.TobaccoUseStatus=FULLY_STOPPED) => BOUNDED(partA.TobaccoLastUsed,
partA.TobaccoUseRemarks);

  CONSTRAINT DataComplete C4 "Risk history questions answered yes must include remarks."
    BOUNDED(rhqTrue.Remarks);

  CONSTRAINT DataComplete C5 "Health history questions answered yes must include
remarks."
    BOUNDED(hhqTrue.Remarks);
/GROUP
```

BOUNDED function takes a list of data fields and returns true or false based on whether all those fields have a value.

Conditional data requirements.

Required Fields Logic

```
CONSTRAINT ReqFields C1 "Required insured party fields."
   REQFIELDS(insuredParty.FirstName, insuredParty.LastName,
                insuredParty.BirthDate, insuredParty.HasValidDL) AND
   (insuredParty.HasValidDL) => REQFIELDS(insuredParty.DLNumber, insuredParty.DLState);

CONSTRAINT ReqFields C2 "Required Part A fields"
   REQFIELDS(partA.TobaccoUseStatus) AND
   (partA.TobaccoUseStatus=FULLY_STOPPED) =>
                REQFIELDS(partA.TobaccoLastUsed, partA.TobaccoUseRemarks);

CONSTRAINT ReqFields C3a "All risk history questions must be answered."
   REQFIELDS(rhq.Response);

CONSTRAINT ReqFields C3b "Remarks must be provided to all risk questions answered yes."
   REQFIELDS(rhqTrue.Remarks);

CONSTRAINT ReqFields C4a "All health history questions must be answered."
   REQFIELDS(hhq.Response);

CONSTRAINT ReqFields C4b "Remarks must be provided to all health questions answered
yes."
   REQFIELDS(hhqTrue.Remarks);
```

Maximal can track all required fields based on stated logic, which can get quite complex.

- Tracking of required fields is important for many form-based applications. As a user fills in the form, fields may become required or not-required dynamically.
  - This is often hard to program on the client side if the logic gets complex.
  - This logic is better delegated to Maximal which can track them and provide them in an API call.
  - Maximal makes it easy to model intricate logic and provide the agility to change these rules anytime.
- Maximal's API method "getRequiredFields" can be called by the front-end app for marking required fields.
  - The option "unknownsOnly=true" can be used to receive only the fields that are required but unspecified.

Summary

- Maximal offers everything a form application needs for a backend.
  - Data model specification.
  - Automatic mapping and persistence to a relational database.
  - Easy specification of data completeness and data validation logic.
  - Easy to use API.
- Ideally suited for building smart intelligent forms
  - Especially if there is some decisioning involved in the process of user's interactions.
  - Offering what-if scenarios.
  - Reflexive questionnaires
  - Dynamic forms.