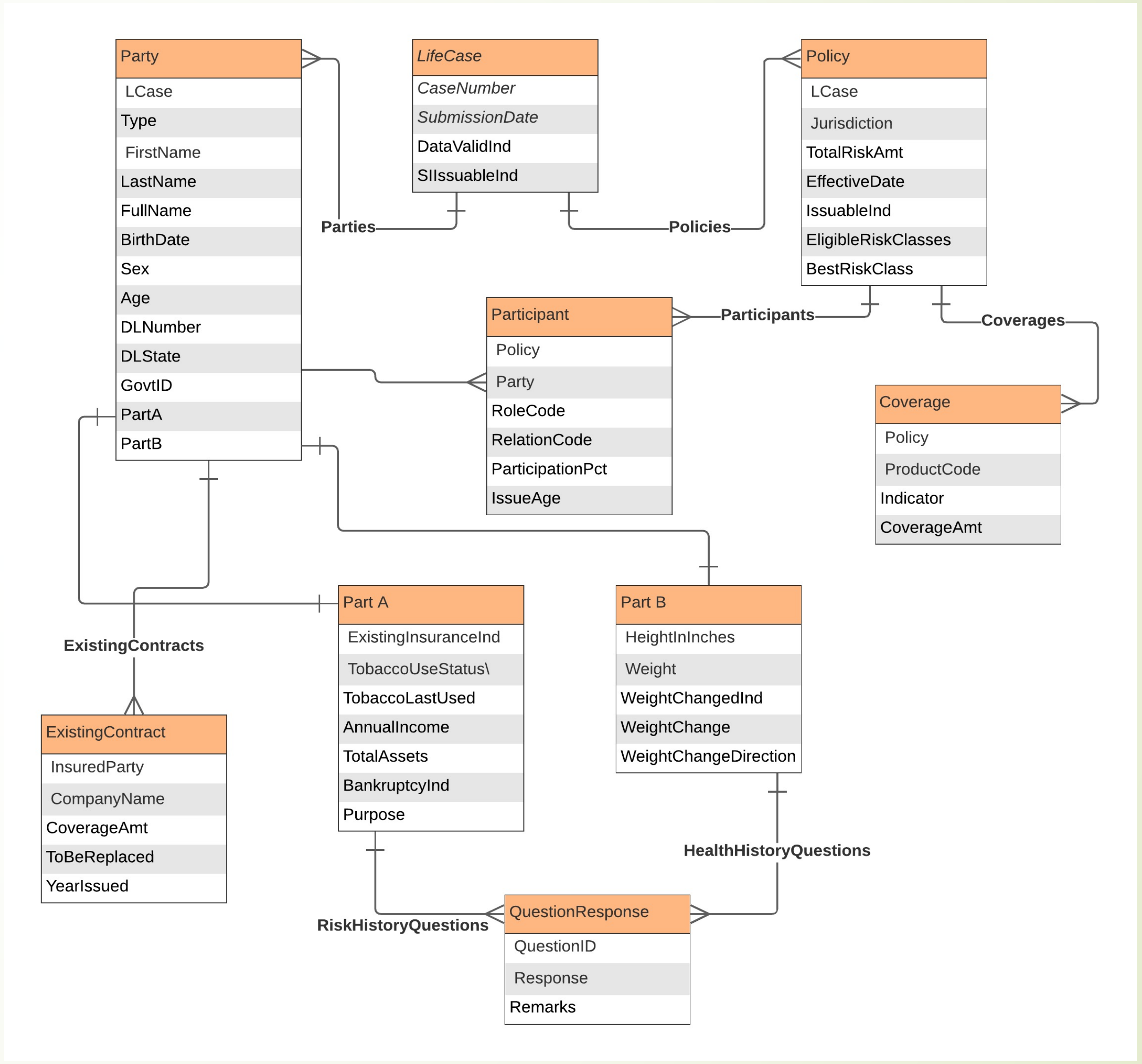
Life Insurance Simplified Issue Eligibility

Introduction

- Simplified Issue in life insurance refers to a process of underwriting simple products automatically and quickly, usually within minutes. This example illustrates a decision model for such underwriting.
- In this decision model, we will do the following;
  - Determine whether the application submitted is qualified for simplified issue.
  - If so, what risk class does it qualify for. Risk class determines premiums.
- We will use criteria based on the data collected from a life insurance application, and for now, ignore the additional data collection that is usually done. Typically, additional data is fetched include MIB (Medical Info Bureau) records, MVR (Motor Vehicle Report), and Rx (prescription data). We can easily extend this model to accommodate those.
- model includes the following modeling concepts:
  - Lookup tables
  - Reference tables
  - Risk class selection by narrowing the choices in a categorical domain (domain reduction)
  - Complex data validation



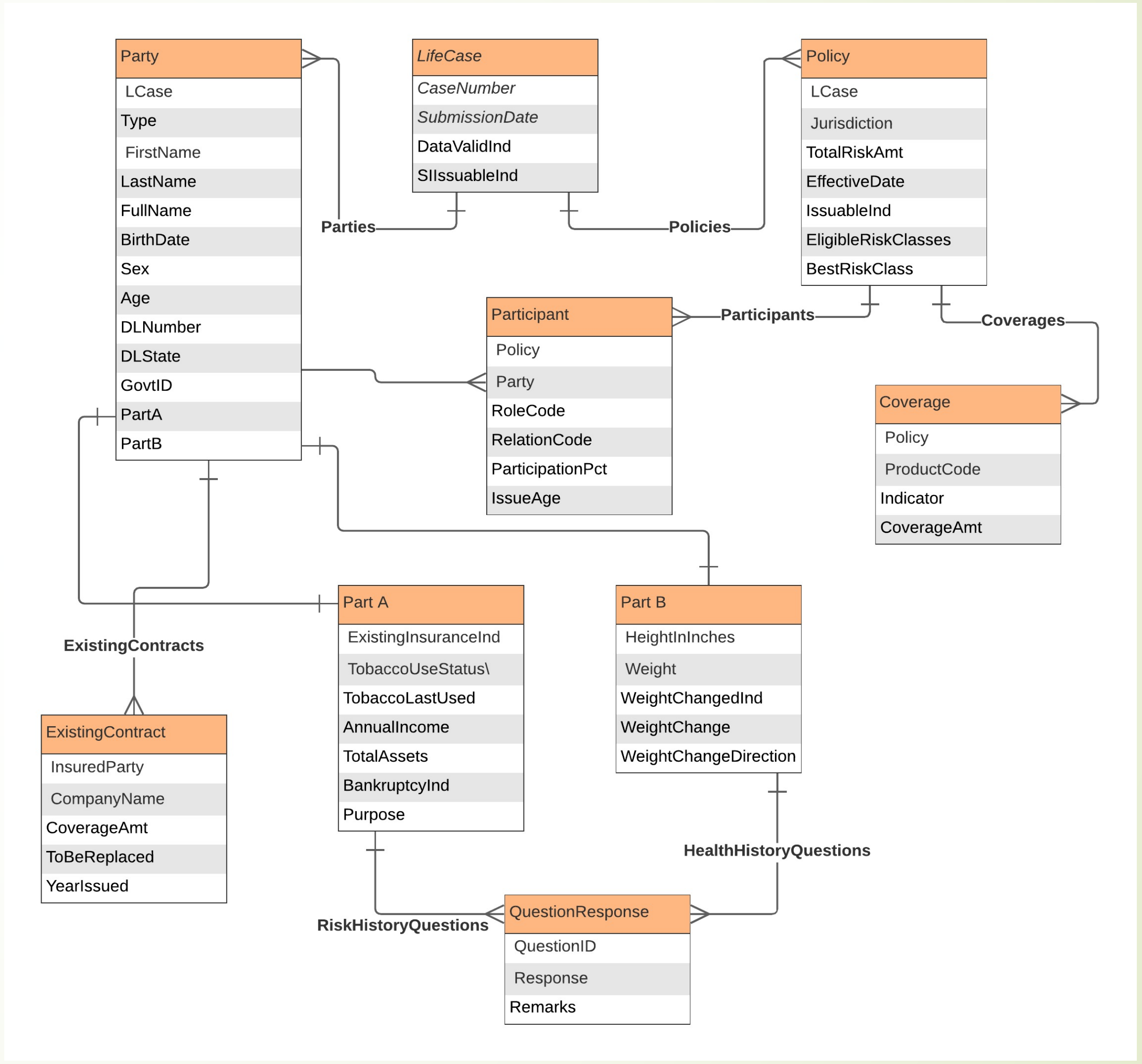
Domain Model

Objects

- .Life Case: represents a case submitted.
- Policy: A life insurance policy applied for. A case may include multiple policies.
- Party: A person, business or a trust that is a part of the case. A party may be the insured risk, a beneficiary, an owner and so on.
- Participant: A policy has multiple participants, recognized by the role code: insured risk, beneficiary, owner, etc.
- Coverage: A policy has multiple coverages, generally one base coverage and zero or more riders.
- Part A: Part A is the application filled up by the applicant / agent collecting risk history, existing insurance, financial information, and so on.
- Part B: This collects medical / health related information from the applicant.
- ExistingContract: Information about existing insurance contracts. Their information is needed both for regulatory purposes and for estimating risk.
- QuestionResponse: Part A and Part B of the application have many questions, usually reflexive, to collect data. They are yes or no questions, with optional remarks.

Relations

The relations in this model are depicted in the diagram.



Decision Logic

maximal

To be eligible for simplified, the case must meet the following conditions:

- Issue age and coverage amounts must be within products specs.
- Data provided in the application must be valid:
  - Beneficiary data must be provided.
  - Beneficiary percentages must total exactly 100%.
  - Contingent beneficiary percentages must total exactly 100%.
- Must meet all simplified issue eligibility
  - No risk history questions answered yes in Part A.
  - No health history questions answered yes in Part B.
  - Coverage amount is less than $500,000.
  - Not a replacement policy.
  - Product must be a Term Life product.
  - The insured must qualify for one of Preferred, Standard, Preferred Smoker and Standard Smoker classes.
- Financial guidelines
  - The ratio of total coverage amount, including all existing unreplaced contracts, to annual income cannot exceed income-factor table.
- Risk Class determination
  - Risk class is determined based on height and weight info from Part A from the build chart.
  - Preferred and Standard classes require insured to be smoke-free for at least five years.

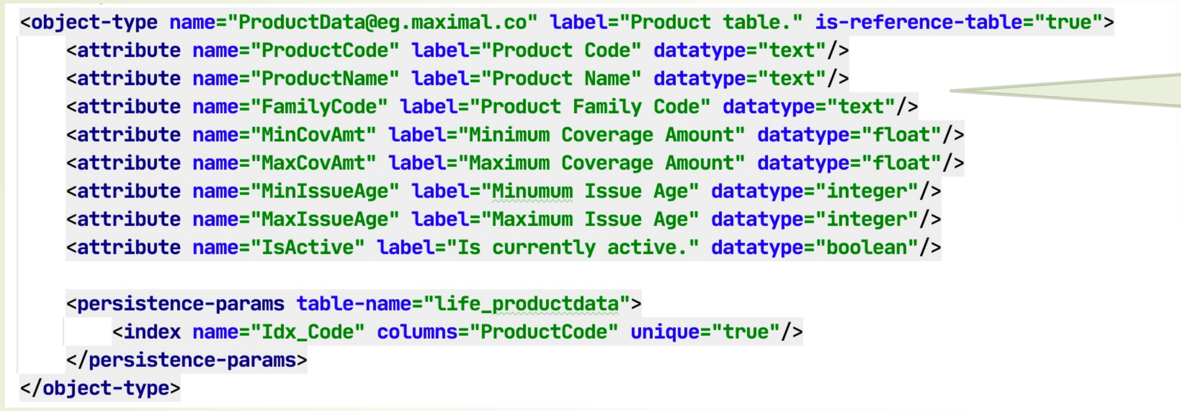
In Maximal, a reference table is used to store mostly static data such as product specification. These reference tables are looked up from the constraint model.

We use the "Group" construct to specify all constraints that are required to be met for a condition to be true.

Income factor table and build chart table are modeled as lookup tables. These lookup tables are looked up from constraints.

Risk class determination requires narrowing of eligible classes based on criteria. We will discuss domain reduction in this context.

Reference Tables



```
<object-type name="ProductData@eg.maximal.co" label="Product table." is-reference-table="true">
    <attribute name="ProductCode" label="Product Code" datatype="text"/>
    <attribute name="ProductName" label="Product Name" datatype="text"/>
    <attribute name="FamilyCode" label="Product Family Code" datatype="text"/>
    <attribute name="MinCovAmt" label="Minimum Coverage Amount" datatype="float"/>
    <attribute name="MaxCovAmt" label="Maximum Coverage Amount" datatype="float"/>
    <attribute name="MinIssueAge" label="Minumum Issue Age" datatype="integer"/>
    <attribute name="MaxIssueAge" label="Maximum Issue Age" datatype="integer"/>
    <attribute name="IsActive" label="Is currently active." datatype="boolean"/>

    <persistence-params table-name="life_productdata">
        <index name="Idx_Code" columns="ProductCode" unique="true"/>
    </persistence-params>
</object-type>
```

ProductData is specified to be a reference table with all corresponding columns. This table can be edited on Maximal developer portal directly.

In a constraint model, declare ProductData as a refrence table with input keys. In this case "ProductCode", which is an attribute of ProductData, is specified as the input key. If there are multiple, they are specified in comma-separated format.

```
REFERENCE_TABLE ProductData = ProductData@eg.maximal.co ON KEYS ProductCode;

CONSTRAINT Product C1 "Face amount cannot exceed product-specific limits."
   policy.IssuableInd =>
   (basecov.CoverageAmt >= ProductData[basecov.ProductCode].MinCovAmt AND
   basecov.CoverageAmt <= ProductData[basecov.ProductCode].MaxCovAmt);

CONSTRAINT Product C2 "Issue age cannot exceed product-specific limits."
   policy.IssuableInd =>
   (insured.IssueAge >= ProductData[basecov.ProductCode].MinIssueAge AND
    insured.IssueAge <= ProductData[basecov.ProductCode].MaxIssueAge);
```

Looking up Min and Max coverage amounts from the reference table by passing basecov.ProductCode as the key in brackets.

Lookup Tables

```
LOOKUP_TABLE IncomeFactorTable "Income factor limits based on age."
   INPUT NUMBER IssueAge,
   OUTPUT NUMBER IncomeFactor,
/* IssueAge      IncomeFactor
--------------------------------*/
   [<=35]         30,
   [36..45]       25,
   [46..50]       20,
   [51..55]       15,
   [56..65]       10,
   [66..70]        5,
   [>=71]          2;

CONSTRAINT Financial C1 "Coverage Amount cannot exceed the income factor for underwriting."
   policy.IssuableInd =>
   (partA.Purpose = INCOME_REPL =>
    policy.TotalRiskAmt <= IncomeFactorTable[insured.IssueAge].IncomeFactor * partA.AnnualIncome);
```

The Lookup Table named IncomeFactorTable is specified with inputs and outputs. A table may have multiple inputs and outputs.

Here we look up the IncomeFactorTable by sending insured.IssueAge as the input. Based on the value of IssueAge, that lookup will return a number value. Using that lookup, we construct a condition for TotakRiskAmt and Annualincome for policy Issuability.

Grouping

maximal

```
GROUP case.SIIssuableInd

   CONSTRAINT SIEligibility C1 "Policy must meet all requirements for Simplified Issue."
      policy.IssuableInd;

   CONSTRAINT SIEligibility C2 "All risk history questions must be answered false for Simplified Issue."
      SSIZE(rhqTrue) = 0;

   CONSTRAINT SIEligibility C3 "All health history questions must be answered false for Simplified Issue."
      SSIZE(hhqTrue) = 0;

   CONSTRAINT SIEligibility C4 "For Simplified Issue, coverage amount must be below $500,000."
      basecov.CoverageAmt < 500000;

   CONSTRAINT SIEligibility C5 "Replaement policies cannot be Simplified Issue."
      SSIZE(replacedContract)=0;

   CONSTRAINT SIEligibility C6 "Only term products are eligible for Simplified Issue.."
      ProductData[basecov.ProductCode].FamilyCode = "TERM";

   CONSTRAINT SIEligibility C7 "Data submitted must be valid for Simplified Issue."
      case.DataValidInd;

   CONSTRAINT SIEligibility C8 "Insured must be eligible for at least one risk class for Simplified Issue."
      policy.BestRiskClass != NONE;

/GROUP;
```

Instead of keep repeating Imples statements, we put it under a group. In this group, if any of the contained constraints are not satisified, case.SIIssuableInd becomes false.

End of group

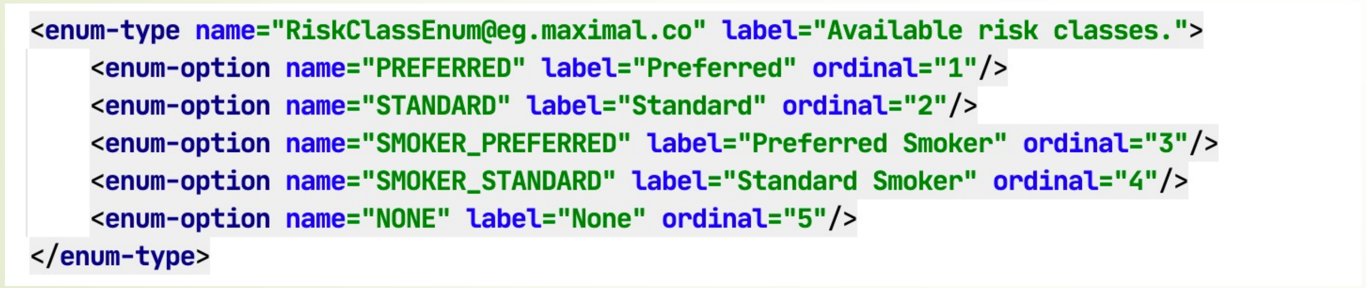
Risk Class Selection / Domain Reduction

```
CONSTRAINT Build C1 "preferred/smoker preferred weight restriction."
   policy.EligibleRiskClasses IN [PREFERRED, SMOKER_PREFERRED] =>
   partB.Weight <= MaxWeightChart[partB.HeightInInches].Preferred;

CONSTRAINT Build C3 "Super standard/smoker preferred weight restriction."
   policy.EligibleRiskClasses IN [STANDARD, SMOKER_STANDARD] =>
   partB.Weight <= MaxWeightChart[partB.HeightInInches].Standard;

CONSTRAINT RiskClass C2 "preferred and standard classes are not offered for smokers."
   policy.EligibleRiskClasses IN [PREFERRED, STANDARD] =>
   (partA.TobaccoUseStatus=NEVER_USED OR
    (partA.TobaccoUseStatus=FULLY_STOPPED AND partA.TobaccoLastUsed=MORE_THAN_5YR));

CONSTRAINT RiskClass C1 STRICT "Best risk class from all the eligible risk classes."
   policy.BestRiskClass = MINOPTION(policy.EligibleRiskClasses);
```

EligibleRiskClasses is of type "multienum". This can hold multiple enumerated values as its value.

Here we are specifying a restriction rule – for EligibleRiskClasses to contain PREFERRED or SMOKER_PREFERRED, the weight should be less than we we look up from the build chart.

If that condition is not satisfied, then these two choices are eliminated from EligibleRiskClasses.



```
<enum-type name="RiskClassEnum@eg.maximal.co" label="Available risk classes.">
    <enum-option name="PREFERRED" label="Preferred" ordinal="1"/>
    <enum-option name="STANDARD" label="Standard" ordinal="2"/>
    <enum-option name="SMOKER_PREFERRED" label="Preferred Smoker" ordinal="3"/>
    <enum-option name="SMOKER_STANDARD" label="Standard Smoker" ordinal="4"/>
    <enum-option name="NONE" label="None" ordinal="5"/>
</enum-type>
```

We assign BestRiskClass to the option available that has the minimum ordinal value. The risk class enumeration with options and ordinals are defined in the domain model.

Data Validation

```
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.

Summary

- Maximal offers rich constructs to make modeling easy.
  - Data tables, lookup tables, groups, domain reduction.
- Maximal makes it easy to work with complex relationships in the data model.
  - Thanks to conditional and dynamic set definitions.
- It is easy to extend this model to bring in any other data – just add additional constraints in a separate section for MIB, MVR, Rx and Blood results, and it will work. No need to think about dependencies or ranking; Maximal takes care of it all via constraint logic propagation.
- All the data is persisted, and decision analytics come out of box.
  - With transactional relational persistence behind the scenes, you can start writing full fledged apps with a complex business logic backend with ease.