Capr Basics Tutorial

Contents

The purpose of `Capr` is to provide a programmatic way to create cohorts using R. What separates `Capr` cohort building from what is already possible in ATLAS is:

1. a code based cohort creation API in R
2. ability to reuse component parts of a cohort definition

Component parts capture elements of a cohort definition that can be reused in multiple cohort definitions. For example, suppose we want to conceptualize a medical diagnosis that we want to deploy multiple times within the cohort definition; once as an ‘additional criteria’ and again as a ‘nested criteria’ within an inclusion rule. `Capr` allows the user to create the diagnosis once and reuse it both parts of the cohort definition. Another example is using the same primary criteria across 20 cohorts. The primary criteria component can be saved and reused in cohort development. The goal of `Capr` is to assist and simplify cohort development while maintaining downstream compatibility with the OHDSI Methods Library. By translating the foundation of cohort development established by circe-be into R, we hope that users can leverage the R environment in assisting them in efficiently developing cohorts and expand possibilities of testing phenotype sensitivity.

The objective of this vignette for `Capr` is to:

1. demonstrate creation of a cohort definition within R,
2. deploy (i.e. serialize and generate) the cohort definition using `CirceR`
3. introduce reusable component parts and show how to save and load these constructs,
4. illustrate how to import and modify existing ATLAS cohorts and
5. use cohort building calls as data in R.

0.1 Creating a Cohort Definition in R

0.1.1 Set Up

Before building a cohort with Capr we need to load some R packages and connect to an OMOP CDM. Capr relies on accessibility to an OMOP Vocabulary schema to find concepts and leverage relationships. We hope in the future we can set a public connection to ATHENA or an OMOP Vocabulary database to expand usability. We should note to ensure that you are using the latest OMOP vocabulary when using CAPR.

```
library(Capr)
library(DatabaseConnector)

connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = Sys.getenv("capr_server"),
                                             user = Sys.getenv("capr_user"),
                                             password = Sys.getenv("capr_password"))

connection <- connect(connectionDetails)

vocabularyDatabaseSchema <- Sys.getenv("capr_schema")
```

In this example we are looking to re-build a cohort developed for the COVID-19 study-a-thon (ID:1775485) This cohort was identifying all inpatient visits for COVID-19 patients. This is a description of the cohort:

Cohort Entry Events

People may enter the cohort when observing any of the following:

1. visit occurrences of 'InpatientVisit', starting after December 1, 2019.

Restrict entry events to with any of the following criteria:

1. having at least 1 condition occurrence of 'COVID-19 (including asymptomatic)', starting 21 days before cohort entry start date and starting anytime on or before cohort entry end date.
2. having at least 1 condition occurrence of any condition (including 'COVID-19 source codes' source concepts), starting 21 days before cohort entry start date and starting anytime on or before cohort entry end date.
3. having at least 1 measurement of 'COVID-19 specific test', starting 21 days before cohort entry start date and starting anytime on or before cohort entry end date.
4. having at least 1 measurement of 'Covid-19 Specific Measurement', starting 21 days before cohort entry start date and starting anytime on or before cohort entry end date; with value as concept: "positive", "detected", "present", "detected", "present" or "positive".
5. having at least 1 observation of 'Covid-19 Specific Measurement', starting 21 days before cohort entry start date and starting anytime on or before cohort entry end date; with value as concept: "positive", "detected", "present", "detected", "present" or "positive".
6. having at least 1 observation of any observation (including 'COVID-19 source codes' source concepts), starting 21 days before cohort entry start date and starting anytime on or before cohort entry end date.

Inclusion Criteria

1. >=18 years old

Entry events with the following event criteria: who are >= 18 years old.

2. Has >= 365 of observation

Entry events having at least 1 observation period, starting anytime up to 365 days before cohort entry start date and ending between 0 days before and all days after cohort entry end date.

3. does not have hospitalization for COVID19 in the 6 months preceding admission

Entry events having no visit occurrences of 'InpatientVisit', starting in the 180 days prior to cohort entry start date; with any of the following criteria:

1. having at least 1 condition occurrence of 'COVID-19 (including asymptomatic)', starting 21 days before 'InpatientVisit' start date and starting anytime on or before 'InpatientVisit' end date.
2. having at least 1 condition occurrence of any condition (including 'COVID-19 source codes' source concepts), starting 21 days before 'InpatientVisit' start date and starting anytime on or before 'InpatientVisit' end date.
3. having at least 1 measurement of 'COVID-19 specific test', starting 21 days before 'InpatientVisit' start date and starting anytime on or before 'InpatientVisit' end date.
4. having at least 1 measurement of 'Covid-19 Specific Measurement', starting 21 days before 'InpatientVisit' start date and starting anytime on or before 'InpatientVisit' end date; with value as concept: "positive", "detected", "present", "detected", "present" or "positive".
5. having at least 1 observation of 'Covid-19 Specific Measurement', starting 21 days before 'InpatientVisit' start date and starting anytime on or before 'InpatientVisit' end date; with value as concept: "positive", "detected", "present", "detected", "present" or "positive".
6. having at least 1 observation of any observation (including 'COVID-19 source codes' source concepts), starting 21 days before 'InpatientVisit' start date and starting anytime on or before 'InpatientVisit' end date.

Limit qualifying entry events to the all events per person.

Cohort Exit

The cohort end date will be offset from index event's end date plus 0 days.

Cohort Eras

Entry events will be combined into cohort eras if they are within 0 days of each other.

0.1.2 Looking up Concepts

The first thing to do in building a cohort is look up clinical concepts we want to use in the cohort. In the example below we want to lookup Inpatient Visits an Emergency Room and Inpatient Visit and an Inpatient Visit. With an existing database connection we can lookup some standard concept Ids in the vocabulary table.

```r
# ConceptId 262 = Emergency Room and Inpaitent Visit
# Concept Id 9201 = Inpatient visit
getConceptIdDetails(conceptIds = c(262, 9201),
                    connection = connection,
                    vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                    mapToStandard = TRUE)
```

Say we do not have a standard OMOP concept id that we want to query but a code from a medical vocabulary like ICD10CM. We can look up this concept code and find the non-standard concept ID.

```r
getConceptCodeDetails(conceptCode = "E11",
                      vocabulary = "ICD10CM",
                      connection = connection,
                      vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                      mapToStandard = FALSE)
```

If we only know the ICD10CM code but want the associated standard concept we can tell our function to `mapToStandard`. In this case the standard SNOMED concept is returned.

```r
getConceptCodeDetails(conceptCode = "E11",
                      vocabulary = "ICD10CM",
                      connection = connection,
                      vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                      mapToStandard = TRUE)
```

Finally if we only know the medical concept but not a specific code we can do a keyword search to find any concepts that contain a character string of "Diabetes".

```r
Diabetes <- lookupKeyword(keyword = "Diabetes",
                          searchType = "any",
                          connection = connection,
                          vocabularyDatabaseSchema = vocabularyDatabaseSchema)

head(Diabetes)
```

0.1.3 Working with Concept Set Expressions

Next we want to take this concept set and turn it into a concept set expression. The concept set expression contains the concept set item which sets how we want to use the concept. We can find descendant concept, exclude it from the list or include its mapped concepts to the expression. In `Capr` when a concept set expression is created the default is to include descendants. When we create the concept set expression we generate a global unique identifier for this component. In the OMOP cohort concept sets are referred to internally using an integer starting from 0. This identifier deploys the concept set expression within a component. With `Capr` we want these pieces to exist in isolation outside the cohort definition so we don't have to rely on this arbitrary number within the cohort definition. This allows us to re-purpose any concept set expression and parent component for a new cohort. In the structural print we see the guid in the concept set expression section of the component part.

```r
IpVisitCSE <- getConceptIdDetails(conceptIds = c(262, 9201),
                                  connection = connection,
                                  vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                  mapToStandard = TRUE) %>%
  createConceptSetExpression(Name = "InpatientVisit", includeDescendants = TRUE)
```

Returning to the idea of concept set items, we can customize the mapping of the concept set item. If we had three concepts in the set but only want to exclude one of them we can specify the position in the `createConceptMapping` function.

QUESTION: Could this use case be handled by createConceptSetExpression.

```r
cid <- c(37310282L, 37310281L, 756055L)
nm <- "COVID-19 specific testing (pre-coordinated Measurements excluded)"
n <- length(cid)

# lookup up covid19 meausres
MeasureOfCovid19 <- getConceptIdDetails(conceptIds = cid,
                                        connection = connection,
                                        vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                        mapToStandard = FALSE)

# create a custom mapping list
```

```
conceptMapping <- createConceptMapping(n = n,
                                       includeDescendants = rep(TRUE, n),
                                       isExcluded = c(TRUE, TRUE, FALSE))

MeasureOfCovid19CSE <- MeasureOfCovid19 %>%
  createConceptSetExpressionCustom(Name = nm, conceptMapping = conceptMapping)
```

0.1.4 Building a Query

Queries look up a concept set in a domain table in the CDM a returns the set of persons in the table that satisfy this condition. A query contains a concept set expression and a list of one or more attributes, which can be NULL. In this example we create an occurrence start date attribute. The create attribute function names are of the form `create_____Attribute()` where the name of the attribute we wish to create is in the blank space. We can look up attributes using the command `listAttributeOptions`. In this example we start by building an occurrence start date attribute. We want the occurrence of the visit to take place any time after December 1st 2019.

```
DateAtt <- createOccurrenceStartDateAttribute(Op = "gt", Value = "2019-12-01")
```

Next we create a visit occurrence query for the inpatient concept set expression and date attribute. Notice at every point we are building up our component container.

```
IpVisitQuery <- createVisitOccurrence(conceptSetExpression = IpVisitCSE,
                                      attributeList = list(DateAtt))
```

This query is what we use for the primary criteria of this cohort. All that remains, is adding an observation window and a limit (number of observations per person) to set the cohort entry. Notice again in the structural print the component inherits the query and evolves into a primary criteria component class. If one ever wants to check the component class they can use `componentType(pc)`.

```
PcCovidDiag <- createPrimaryCriteria(Name = "Inpatient Visit Primary Criteria",
                              ComponentList = list(IpVisitQuery),
                              ObservationWindow = createObservationWindow(0L,0L),
                              Limit = "All")
```

0.1.5 Creating a Count

Count classes inherit from queries and add Boolean logic and temporal constraints. This class counts the number of occurrences of a query that fit within an observation period relative to the initial event. Counts are used in cohort restriction like in the additional criteria and inclusion rules. In this example we are going to build a count for COVID-19. We first look up the concept and include descendants in the mapping. Next we make the concept set expression into a query without any attributes. A count requires a timeline so we can create a time window relative to the initial event.

```
#lookup covid diagnosis in vocabulary
CovidDiag <- getConceptIdDetails(conceptIds = 37311061,
                                 connection = connection,
                                 vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                 mapToStandard = TRUE)

#create concept set expression including descendant concepts
```

```r
CovidDiagCSE <- CovidDiag %>%
  createConceptSetExpression(Name = "COVID-19 (including asymptomatic)",
                             includeDescendants = TRUE)

# create a conditionOccurrence query
CovidDiagQuery <- createConditionOccurrence(conceptSetExpression = CovidDiagCSE)

#create start window for timeline
#start window to count occurrences from inpatient event (cohort entry)
# 21 days before IpVisit index start to all days after IpVisit index start
StartWindow1 <- createWindow(StartDays = 21, StartCoeff = "Before",
                             EndDays = "All", EndCoeff = "After")

#create end window for timeline
#end window to count occurrences from inpatient event (cohort entry)
#end all days before IpVisit index start
#0 days after IpVisit index start
EndWindow1 <- createWindow(StartDays = "All", StartCoeff = "Before",
                           EndDays = 0, EndCoeff = "After",
                           IndexStart = FALSE) #toggle index end

#create timeline
Timeline1 <- createTimeline(StartWindow = StartWindow1,
                            EndWindow = EndWindow1)

#create a count criteria
#at least 1 occurrence of a covid diagnosis occurring
#between a) 21 days before and all days after initial inpatient visit index start date
#and b) all days before and 0 days after initial inpatient visit index end date
CovidCountCriteria1 <- createCount(Query = CovidDiagQuery,
                                   Logic = "at_least",
                                   Count = 1,
                                   Timeline = Timeline1)
```

0.1.6 Creating a Group

A group is a set of counts, queries, attributes and sub-groups that are deployed using boolean logic. We can combine multiple components and then describe if the patient needs to satisfy all these criteria to remain in the cohort or at least 1. A group is used for additional criteria and inclusion rules. It is also used for a nested criteria attribute. In this example we will construct multiple counts that comprise of a group. Within this section we also show sub-examples of nuances of `Capr` as we build up to the group.

0.1.6.1 Creating a Source Concept Attribute In the code chunk below we want to use source concepts as an attribute for one of the criteria in the group. The patient must have at least 1 instance of these source concepts, excluding for 45542411, 586414, 45600471, 586415, 710157. We first create a concept mapping for the length of the concept set and then toggle the mapping so that these five concepts have the item of isExcluded set to TRUE. Once the conceptMapping is set we can create the concept set Expression. The concept set expression known as COVID SourceConcept can be used as a source concept attribute. Notice when we make the query in the `createConditionOccurrence` function we do not include the concept set expression like before. This is because we are making a special attribute for the source concepts. We should note that a query does not always contain a concept set expression. This is also the case when making an observation period. One should be careful on how they wish to apply the concept set expressions within the query, it may be an attribute.

```r
CovidSourceConcepts <- getConceptIdDetails(conceptIds = c(710158L, 710155L, 710156L,
                                                          710159L, 45542411L, 710160L,
                                                          45756093L, 42501115L, 586414L,
                                                          45600471L, 586415L, 710157L),
                                           connection = connection,
                                           vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                           mapToStandard = FALSE)
#create a default mapping list includeDescendants, isExcluded, includeMapped all FALSE
ConceptMappingSourceConcepts <- createConceptMapping(n = nrow(CovidSourceConcepts))

#toggle TRUE the positions needed to alter
ConceptMappingSourceConcepts <- toggleConceptMapping(conceptMapping = ConceptMappingSourceConcepts,
                                                     pos = c(5,9:12), mapping = "isExcluded")

#Create Concept Set Expression with custom mapping
CovidSourceConceptCSE <- CovidSourceConcepts %>%
  createConceptSetExpressionCustom(Name = "COVID-19 source codes",
                                   conceptMapping = ConceptMappingSourceConcepts)

#Create a Source Concept Attribute for query
SourceConceptAttribute <- createConditionSourceConceptAttribute(ConceptSetExpression = CovidSourceConcep

#Create Query
CovidSourceConceptQuery <- createConditionOccurrence(attributeList = list(SourceConceptAttribute))

#create Count for covid source concepts
CovidCountCriteria2 <- createCount(Query = CovidSourceConceptQuery,
                                   Logic = "at_least",
                                   Count = 1,
                                   Timeline = Timeline1)
```

0.1.6.2 Forward Piping and Mutability of the Component Class Something you probably have noticed is most of `Capr` code contains forward pipes from the `magrittr` package in R. This usually looks a little nicer to avoid overuse of the assignment operator `<-`. However we use this deliberately to convey the mutability of the component container in `Capr`. Looking back at the structural prints from before we can see that component class changes as the object moves from a concept set expression, to a query, to a count and so forth. However the information from the previous state stays the same. The S4 component allows us to modify the structure while maintaining some rigid consistency across inheritance. In the code below, our look up returns a data.frame. The input for the `createConceptSetExpression` is a data.frame and the output is component object with a component class of conceptSetExpression. Next we mutate this component to build a query. On can always check the current component class of a component by using the function `componentType`

```r
CovidCountCriteria3 <- getConceptIdDetails(conceptIds = 37310282,
                                 connectionDetails = NULL,
                                 connection = connection, #use connection since it is already open
                                 vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                 oracleTempSchema = NULL,
                                 mapToStandard = TRUE) %>%
  createConceptSetExpression(Name = "COVID-19 specific test") %>%
  createMeasurement(conceptSetExpression = .) %>%
  createCount(Logic = "at_least",
              Count = 1,
              Timeline = Timeline1)
componentType(CovidCountCriteria3)
```

0.1.6.3 Creating a Concept Attribute

Sometimes we want to apply concepts directly into a cohort as an attribute, without building them as a concept set expression in the cohort. In this case we are using different values as concepts for measurement of a COVID-19 test. Below we show a lookup as before for concept values.

```
getConceptIdDetails(conceptIds = c(4126681L,45877985L, 9191L, 4181412L, 45879438L, 45884084L),
                    connection = connection,
                    vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                    mapToStandard = TRUE)
```

If we wanted to use these concepts directly into an attribute we can search them from the function call. Underlying this function is the same lookup command as before in addition to some formatting changes for concepts that are incorporated within the cohort and not in the concept set list.

```
ValueAsConceptAtt <- createValueAsConceptAttribute(conceptIds = c(4126681L,45877985L, 9191L,
                                                                   4181412L, 45879438L, 45884084L),
                                                   connectionDetails = NULL,
                                                   connection = connection, #use connection since it is
                                                   vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                                   oracleTempSchema = NULL,
                                                   mapToStandard = TRUE)

Covid19MeasuresQuery <- createMeasurement(conceptSetExpression = MeasureOfCovid19CSE,
                                          attributeList = list(ValueAsConceptAtt))
CovidCountCriteria4 <- createCount(Query = Covid19MeasuresQuery,
                                   Logic = "at_least",
                                   Count=1, Timeline = Timeline1)
```

0.1.6.4 Reusing Component Parts

`Capr` leverages the global environment in R to maintain objects that have been previously created in other parts of the cohort. In `Capr`, components and other objects can always exist in isolation so we can simply add them into a different function call. Notice that our object `Timeline1` which represents the timeline of the restricted events has stayed the same for each new count. We are recycling the object from the global environment and do not need to recreate this every time. We will expand upon this idea in the save and load commands in `Capr`. Beyond reuse we can copy and modify aspects to generate new component objects. In the code chunk below we can build this count through our typical means: lookup the concept, create the mapping, make the concept set expression, set the query with attributes and then build the count.

```
Covid19ObservationMeasureQuery <- createObservation(conceptSetExpression = MeasureOfCovid19CSE,
                                          attributeList = list(ValueAsConceptAtt))

CovidCountCriteria5A <- createCount(Query = Covid19ObservationMeasureQuery,
                                   Logic = "at_least",
                                   Count = 1,
                                   Timeline = Timeline1)
```

This is nearly identical to something we have created before, except it was a Measurement domain. Instead of recreating everything we can use the copy and modify principles in R to more quickly change this component to the new domain observation.

```
CovidCountCriteria5 <- CovidCountCriteria4
CovidCountCriteria5@CriteriaExpression[[1]]@Criteria@Domain <- "Observation"
```

0.1.6.5 Finally, Creating the group

We need to make one more count for our group. Once this is created we can add all these components into a group. A group requires a name, some logic on what counts and the list of criteria, demographic criteria and sub groups.

```
CovidSourceConceptQueryObs <- createObservation(attributeList = list(
  createObservationSourceConceptAttribute(CovidSourceConceptCSE)))
CovidCountCriteria6 <- createCount(Query = CovidSourceConceptQueryObs,
                          Logic = "at_least",
                          Count =1,
                          Timeline = Timeline1)

CovidDiagGroup <- createGroup(Name = "COVID-19 Diag",
                          type = "ANY",
                          criteriaList = list(CovidCountCriteria1, CovidCountCriteria2,
                                              CovidCountCriteria3, CovidCountCriteria4,
                                              CovidCountCriteria5, CovidCountCriteria6))
saveComponent(CovidDiagGroup,
              saveName = "CovidDiagGroup",
              savePath = "~/Documents")
```

We can use this group to make an additional criteria like in the chunk below.

```
AcCovidDiag <- createAdditionalCriteria(Name = "Additional Crit for COVID cohort",
                               Contents = CovidDiagGroup,
                               Limit = "All")
```

We can also use this same group as a nested (correlated) criteria for an inclusion rule. This is another example of the resusbility of the components. We can make a group once and then use it in a variety of scenarios across the cohort definition. We can also save the group and use it another time in a different cohort if we plan to use it frequently across a family of like cohorts.

```
#load an existing component
CovidDiagGroupLoad <- loadComponent("~/Documents/CovidDiagGroup.json")
CorrelatedCriteriaAtt <- createCorrelatedCriteriaAttribute(CovidDiagGroupLoad)
InpatientVisitWCovidDiagQuery <- createVisitOccurrence(conceptSetExpression = IpVisitCSE,
                                    attributeList = list(CorrelatedCriteriaAtt))
Timeline2 <- createTimeline(StartWindow = createWindow(StartDays = 180, StartCoeff = "Before",
                                                EndDays = 1, EndCoeff = "Before"))

CovidCountInclusionRule3 <- createCount(Query = InpatientVisitWCovidDiagQuery,
                            Logic = "exactly",
                            Count = 0,
                            Timeline = Timeline2)
CovidHospitalizationGroup <- createGroup(Name ="does not have hospitalization for COVID19 in the 6 mont
                           type = "ALL",
                           criteriaList = list(CovidCountInclusionRule3),
                           demographicCriteriaList = NULL,
                           Groups = NULL)
```

0.1.6.6 Creating a Demographic Criteria In our groups sometimes we want to include a demographic criteria for examples persons 18 years or older. Demographic criteria are attributes, so we use the attribute functions to create them like before.

```
AgeAtt <- createAgeAttribute(Op = "gte", Value = 18)
Age18AndOlderGroup <- createGroup(Name = ">=18 years old",
                           type="ALL",
                           criteriaList = NULL,
                           demographicCriteriaList = list(AgeAtt),
                           Groups = NULL)
```

0.1.7 Creating Inclusion Rules

Once we have a series of groups we are happy with we can bundle them together to create inclusion rules. The code chunk below shows how to make inclusion rules from the groups we have created before plus a group that is based on persons having at least 365 days of observation.

```
Timeline3 <- createTimeline(StartWindow = createWindow(StartDays = "All", StartCoeff = "Before",
                                                       EndDays = 365, EndCoeff = "Before"),
                      EndWindow = createWindow(StartDays = 0, StartCoeff = "Before",
                                               EndDays = "All", EndCoeff = "After",
                                               IndexStart = FALSE, EventStarts = FALSE))

CovidCountInclusionRule2 <- createCount(Query = createObservationPeriod(),
                            Logic = "at_least",
                            Count =1,
                            Timeline = Timeline3)
Has365DaysObsGroup <- createGroup(Name = "Has >= 365 of observation",
                           type = "ALL",
                           criteriaList = list(CovidCountInclusionRule2))

IrsCovidDiag <- createInclusionRules(Name = "Inclusion Rules for covid Cohort",
                            Contents = list(Age18AndOlderGroup,
                                            Has365DaysObsGroup,
                                            CovidHospitalizationGroup),
                            Limit ="First")
```

0.1.8 Defining the cohort exit

The cohort exit is defined by the end strategy and the censoring criteria. The end strategy by default is end of continuous observation, however we can create two additional strategies to define how persons exit the cohort. First we can use a date offset. The code chunk below shows an example.

```
EsCovidDiag <- createDateOffsetEndStrategy(offset = 0, eventDateOffset = "EndDate")
```

We can aslo create a custom era using a drug exposure to define the end point of a cohort. This example is not a part of the covid cohort but is included for demonstration. We first find a concept like warfarin, create the concept set expression and then apply this within the custom era.

```
WarfarinCSE <- getConceptIdDetails(conceptIds = 1310149,
                                   connectionDetails = NULL,
                                   connection = connection, #use connection since it is already open
                                   vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                   oracleTempSchema = NULL,
                                   mapToStandard = TRUE) %>%
  createConceptSetExpression(Name = "Warfarin", includeDescendants = TRUE)

EndStrategyWithDrug <- createCustomEraEndStrategy(WarfarinCSE,
                                                  gapDays = 3,
                                                  offset = 0)
```

Aside from the end strategy we can also define a censoring criteria as a way that persons exit the cohort. This is another example that is not part of the covid cohort but we include for the purpose of demonstration. Similar to a primary criteria we can build a concept set expression for what will be our censoring event. Then we create a query (we may also add an attribute) that defines the domain of the concept set expression. We can add any queries we wish to add to the censoring criteria to build this component.

```
InfliximabCSE <- getConceptIdDetails(conceptIds = c(937368,937369),
                                   connectionDetails = NULL,
                                   connection = connection, #use connection since it is already open
                                   vocabularyDatabaseSchema = vocabularyDatabaseSchema,
                                   oracleTempSchema = NULL,
                                   mapToStandard = TRUE) %>%
  createConceptSetExpression(Name = "Infliximab", includeDescendants = TRUE)
infliximabQuery <-createDrugEra(conceptSetExpression = InfliximabCSE)
cen <-createCensoringCriteria(Name = "prev inflix", ComponentList =  list(infliximabQuery))
```

0.1.9 Creating a Cohort Era

The last piece of a cohort definition is to add the cohort era. This defines padding between events and censor window of when to begin recording events. Say in a cohort we only we want to look at events after a certain date. All of this is defined in the cohort era. In many cases the cohort era can be composed of default settings (padding of 0 and no censor window) and does not need to to built for the cohort definition.

```
cohortEra <- createCohortEra(LeftCensorDate = "2019-12-17")
```

0.1.10 Creating the cohort definition

Once we are happy with the creation of all the sub-components we can build the cohort definition. This is pretty simple. We infuse the component parts necessary for the cohort definition and create. In the cohort definition we can add meta data about the cohort definition. This includes a name (required), description (optional), author (optional) and the cdm version compatability. The default of the cdm version is ">= 5.0.0". If we had a different variation of this cohort definition, for example the primary criteria for inpatient has a different observation window, we can simply create a separate primary criteria object and then infuse that into a separate cohort definition.

```
desc <- "This cohort counts the number of inpatient visits from patients with COVID-19, this is counted
cd <- createCohortDefinition(Name = "Inpatient COVID-19 Diag",
                             Description = desc,
                             PrimaryCriteria = PcCovidDiag,
                             AdditionalCriteria = AcCovidDiag,
                             InclusionRules = IrsCovidDiag,
                             EndStrategy = EsCovidDiag)
```

0.2 Deploying the Cohort

Once the cohort definition has been created we want to run it against our OMOP cdm. To do so we need to use the `CirceR` package to take the cohort definition and create the ohdisql to run on our local environment. `CirceR` will populate the parameterized sql if one sets the generate options. We need to tell what is the column for the cohort Id in the cohort table of the results schema, the id number in the results schema, the cdm schema, the table where the cohort is to be written, the results Schema where the table sits, the vocabulary schema to access concepts and a toggle to generate statistics on the cohort. We can take the cohort definition created in R and return objects to deploy this cohort across our OMOP system. The function `compileCohortDefinition` takes the R and builds the json that `CirceR` takes to get the cohort description and most importantly the ohdisql. `Capr` does some low level manipulation to get from the R object to the json and then relies on `CirceR` to build the ohdisql. Functions leveraged in the `compileCohortDefinition` functions from `CirceR` include `cohortExpressionFromJson` which activates the circe-be, `cohortPrintFriendly` which returns the enlish text of the cohort, and `buildCohortQuery` which creates the ohdisql from the input generate options.

```
genOp <- CirceR::createGenerateOptions(cohortIdFieldName = "cohort_definition_id",
                                cohortId = 9999,
                                cdmSchema = vocabularyDatabaseSchema,
                                targetTable = "cohort",
                                resultSchema = "results",
                                vocabularySchema = vocabularyDatabaseSchema,
                                generateStats = F)
cohortInfo <- compileCohortDefinition(cd, genOp)
```

The cohortInfo object in the chunk above is a list of 4 aspects: 1) the s3 cohort definition (`circeS3`) – this is a intermediate structure for cohort definition that one can browse to debug errors 2) the cohort json (`circeJson`) – this returns the json of the cohort definition, identical to ATLAS. You can copy an paste this json to ATLAS to generate the same cohort. You can also save this for record keeping 3) the cohort description (`cohortRead`) – this returns the english text of the cohort 4) the sql (`ohdisql`) – the parameterized sql already filled in with the generate options.

To execute the ohdisql one needs to use the `DatabaseConnector::executeSql` function to run the sql against the db as shown in the chunk before.

```
# TODO Need to improve sql generation. I'm not sure parameters like schema names should be automaticall
sql <- cohortInfo$ohdiSQL
# There are no parameters to be filled in
# What if I want to use a temp emulation schema or execute on a different database?

# cat(sql)
# stringr::str_extract_all(sql, "@\\w+")

# but the sql does need to be rendered because there is branching logic that needs to be normalized

sql <- SqlRender::render(sql)
sql <- SqlRender::translate(sql, "postgresql")
# readr::write_lines(sql, here::here("sql.txt"))
# dbExecute(connection, "ROLLBACK;")
# dbExecute(connection, "DROP TABLE codesets;")
# dbExecute(connection, "DROP TABLE qualified_events;")
# dbExecute(connection, "DROP TABLE inclusion_events;")

DatabaseConnector::executeSql(connection = connection, sql = sql)
```

0.3 Saving and Loading Cohorts

An important feature of `Capr` is the ability to save and load components. As mentioned before we may want to reuse a component in a different cohort. Instead of rebuilding this object we can save the component and load it up to deploy on a different cohort. The idea here is to leverage the programatic environment of R to assist in cohort development and programming network studies. We encourage users to save and recycle components to assist them in study development. The code chunk below shows an example of saving a cohort additional criteria.

```
saveComponent(AcCovidDiag,
              saveName = "covidAC",
              savePath = "~/Documents/ComponentLibrary/CovidInpatient")
```

We can then load this additional criteria into the global environment to utilize in a new iteration of a covid cohort. Maybe this new cohort has a different primary criteria but the same in other places. Notice the saved components are written to json. Future releases may allow saving the S4 object as a binary, depending on user preference.

```
acCovidIp <- loadComponent("~/Documents/ComponentLibrary/CovidInpatient/covidAC.json")


cd2 <- createCohortDefinition(Name = "DifferentcovidDiagCohort",
                              PrimaryCriteria = pcCovidOw365,
                              AdditionalCriteria = acCovidIp,
                              InclusionRules = irs,
                              EndStrategy = es)
```

0.4 Importing existing JSON Cohorts

The OHDSI community has created thousands of cohorts and has introduced a gold standard pheontype library, maintaining a list of ideal cohorts for medical constructs. We do not want to rewrite good existing cohorts. We also may want to take an existing cohort skeleton and make a slight modification. `Capr` allows us to import Circe Json and build the cohort definition to your global environment. With this cohort uploaded you can modify as you see fit. In this example we upload the cohort definition and a saved primary criteria. We modify the primary criteria to have an observation window of post days 365 and add this to our loaded cohort. We can build the ohdisql of this cohort quite quickly.

```
cd2 <- readInCirce("~/Documents/NetworkStudy/COVID/inpatientCovid.json")
pc2 <- loadComponent("~/Documents/ComponentLibrary/CovidInpatient/covidPC.json")
pc2@CriteriaExpression$ObservationWindow@PostDays <- 365L
cd2@PrimaryCriteria <- pc2
cohortInfo2 <- compileCohortDefinition(cd2, genOp)
```

0.5 Cohort Building Language as Data

A cool feature of R is its ability to do metaprogramming. We can save code as data and modify the code programatically. For those interested on learning the theoretical aspects of this in R, one should read Hadley Wickhams's Advanced R section 4 about Metaprogramming. `Capr` uses this feature when importing a CIRCE json. It creates structural calls in an execution environment and then evaluates them into the global environment, leaving only the cohort definition as its trace. `Capr` has a function that places the function calls into a txt file.

```r
writeCaprCall(jsonPath = "~/Documents/NetworkStudy/COVID/inpatientCovid.json",
              txtPath = "~/Documents/NetworkStudy/COVID/inpatientCovid.txt")
```

While available to the user this function is really designed to interact with a webAPI, writing the machine legible R code to generate acohort. This is an area of future exploration to modify cohort definitions programatically using the R calls saved as data.