

Using CohortIncidence

Christopher Knoll

2026-08-20

Contents

1	Introduction	1
2	Installation instructions	1
3	Database Preparation	1
4	A simple example	3
4.1	Build the design	3
4.2	Using age, gender and start year strata	5
4.3	Using executeAnalysis()	5
5	Advanced Usage: Budling and Executing SQL manually	6
5.1	Build analysis SQL from design	6
5.2	Render SQL with paramaters and execute	14
5.3	Using Temp Tables	15

#> Loading required package: DatabaseConnector

1 Introduction

This vignette describes how to use the `CohortIncidence` package to perform a single incidence rate analysis for a given target and outcome cohort, with a few settings for Time At Risk and Clean Window.

2 Installation instructions

Before installing the `CohortIncidence` package make sure you have Java available. Java can be downloaded from www.java.com. For Windows users, RTools is also necessary. RTools can be downloaded from CRAN.

The `CohortIncidence` package is currently maintained in a Github repository.

```
install.packages("remotes")
remotes::install_github("ohdsi/CohortIncidence")
```

Once installed, you can type `library(CohortIncidence)` to load the package.

3 Database Preparation

The results of the anlysis SQL will assume a final table: `@results_database_schema.incidence_summary`. The DDL for this table can be fetched from the package via the following:

```

# Fetch DDL from package
ddl <- CohortIncidence::getResultsDdl()
cat(ddl)
CREATE TABLE @schemaName.incidence_summary
(
    ref_id int,
    source_name varchar(255),
    target_cohort_definition_id bigint,
    tar_id bigint,
    subgroup_id bigint,
    outcome_id bigint,
    age_group_id int,
    gender_id int,
    gender_name varchar(255),
    start_year int,
    persons_at_risk_pe bigint,
    persons_at_risk bigint,
    person_days_pe bigint,
    person_days bigint,
    person_outcomes_pe bigint,
    person_outcomes bigint,
    outcomes_pe bigint,
    outcomes bigint,
    incidence_proportion_p100p float,
    incidence_rate_p100py float
);

CREATE TABLE @schemaName.target_def
(
    ref_id int,
    target_cohort_definition_id bigint,
    target_name varchar(255)
);

CREATE TABLE @schemaName.outcome_def
(
    ref_id int,
    outcome_id bigint,
    outcome_cohort_definition_id bigint,
    outcome_name varchar(255),
    clean_window bigint,
    excluded_cohort_definition_id bigint
);

CREATE TABLE @schemaName.tar_def
(
    ref_id int,
    tar_id bigint,
    tar_start_with varchar(10),
    tar_start_offset bigint,
    tar_end_with varchar(10),
    tar_end_offset bigint

```

```
);

create table @schemaName.age_group_def
(
  ref_id int,
  age_group_id int NOT NULL,
  age_group_name varchar(50) NOT NULL,
  min_age int NULL,
  max_age int NULL
);

CREATE TABLE @schemaName.subgroup_def
(
  ref_id int,
  subgroup_id bigint,
  subgroup_name varchar(255)
);
```

Using `SqlRender` and `DatabaseConnector`, you can execute the above on your target database platform in order to deploy the table. Remember to replace `@schemaName` with the appropriate schema. You can also ‘hack’ the `@schemaName` paramater to apply a prefix by specifying the `SqlRender` paramater of `schemaName.incidence_summary` to a target table ie: `mySchema.prefix_incidence_summary`. Using the same paramater name/value in `buildQuery()` will allow you to provide a prefix to the result table name instead of having to declare a separate schema.

```
connectionDetails <- DatabaseConnector::createConnectionDetails(dbms = "postgresql",server={Sys.getenv(

# to specify the target schema (the typical use case):
ddl <- SqlRender::render(CohortIncidence::getResultsDdl(), schemaName = "mySchema")

# a work-around to provide a prefix to the result table, in case creating new schema is restricted
ddlPrefix <- SqlRender::render(CohortIncidence::getResultsDdl(), "schemaName.incidence_summary" = "mySchema")

con <- DatabaseConnector::connect(connectionDetails)
DatabaseConnector::executeSql(ddl)
DatabaseConnector::disconnect(con)
```

4 A simple example

This example will create a `CohortIncidence` design containing a single target, outcome, and time at risk.

4.1 Build the design

The following script builds a single T, O and Time at Risk, and assembles those element into a design. Finally, the resulting JSON is printed.

```
t1 <- CohortIncidence::createCohortRef(id=1, name="Target cohort 1")

o1 <- CohortIncidence::createOutcomeDef(id=1,name="Outcome 1, 30d Clean",
                                         cohortId =2,
                                         cleanWindow =30)

tar1 <- CohortIncidence::createTimeAtRiskDef(id=1,
```

```

        startWith="start",
        endWith="end",
        endOffset=30)

# Note: c() is used when dealing with an array of numbers,
# later we use list() when dealing with an array of objects
analysis1 <- CohortIncidence::createIncidenceAnalysis(targets = c(t1$id),
                                                    outcomes = c(o1$id),
                                                    tars = c(tar1$id))

subgroup1 <- CohortIncidence::createCohortSubgroup(id=1, name="Subgroup 1", cohortRef = createCohortRef

# Create Design (note use of list() here):
irDesign <- CohortIncidence::createIncidenceDesign(targetDefs = list(t1),
                                                    outcomeDefs = list(o1),
                                                    tars=list(tar1),
                                                    analysisList = list(analysis1),
                                                    subgroups = list(subgroup1))

# Render the design as JSON
irDesign$asJSON(pretty = T)
#> {
#>   "targetDefs": [
#>     {
#>       "id": 1,
#>       "name": "Target cohort 1"
#>     }
#>   ],
#>   "outcomeDefs": [
#>     {
#>       "id": 1,
#>       "name": "Outcome 1, 30d Clean",
#>       "cohortId": 2,
#>       "cleanWindow": 30
#>     }
#>   ],
#>   "timeAtRiskDefs": [
#>     {
#>       "id": 1,
#>       "start": {
#>         "dateField": "start",
#>         "offset": 0
#>       },
#>       "end": {
#>         "dateField": "end",
#>         "offset": 30
#>       }
#>     }
#>   ],
#>   "analysisList": [
#>     {
#>       "targets": [1],
#>       "outcomes": [1],

```

```

#>       "tars": [1]
#>     }
#>   ],
#>   "subgroups": [
#>     {
#>       "CohortSubgroup": {
#>         "id": 1,
#>         "name": "Subgroup 1",
#>         "cohort": {
#>           "id": 300
#>         }
#>       }
#>     }
#>   ]
#> }

```

4.2 Using age, gender and start year strata

The IR design can also include settings to specify if an analysis should be done at the age, gender or start year levels (or any combination of those choices). To use this function, you create the strata settings with the `CohortIncidence::createStrataSettings` function:

```

irDesignWithStrata <-
  CohortIncidence::createIncidenceDesign(
    targetDefs = list(t1),
    outcomeDefs = list(o1),
    tars = list(tar1),
    analysisList = list(analysis1),
    subgroups = list(subgroup1),
    #add by age and by gender strata, but don't do by start year.
    strataSettings = CohortIncidence::createStrataSettings(
      byGender = T,
      byAge = T,
      ageBreaks = list(17, 34, 65),
      ageBreakList = list(list(25), list(65))
    )
  )

```

In the above example, there are 2 ways of specifying the age breaks: `ageBreaks` and `ageBreakList`. `ageBreaks` creates a single age break specification, while `ageBreakList` allows you to specify a list of breaks. All breaks defined in `ageBreaks` and `ageBreakList` will be used if specified. If `byAge` is `TRUE`, you must specify at least one age break specification either in `ageBreaks` or `ageBreakList`.

4.3 Using executeAnalysis()

If there is no need to see the analysis sql or control the output of the analysis to a permanent table, the `executeAnalysis()` function can be used to perform the cohort incidence using a simple API:

```

buildOptions <- CohortIncidence::buildOptions(cohortTable = "demoCohortSchema.cohort",
                                              cdmDatabaseSchema = "mycdm",
                                              sourceName = "mysource",
                                              refId = 1)

executeResults <- CohortIncidence::executeAnalysis(connectionDetails = connectionDetails,

```

```
incidenceDesign = irDesign,
buildOptions = buildOptions)
```

`executeAnalysis()` will return a list of dataframes with the following fields: - incidenceSummary - targetDef - outcomeDef - tarDef - ageGroupDef - subgroupDef

These dataframes follow the same structure as the corresponding tables described in the Database Preparation section.

5 Advanced Usage: Building and Executing SQL manually

There may be a reason (debugging or special processing steps) Where you would want to access the analysis SQL before execution.

The following sections describe how to fetch the SQL, translate and execute the statements.

5.1 Build analysis SQL from design

From the previous design, the `CohortIncidence::buildQuery()` method is used to generate the analysis SQL:

```
buildOptions <- CohortIncidence::buildOptions(cohortTable = "demoCohortSchema.cohort",
                                              cdmDatabaseSchema = "mycdm",
                                              resultsDatabaseSchema = "myresults",
                                              sourceName = "mysource",
                                              refId = 1)

analysisSql <- CohortIncidence::buildQuery(incidenceDesign = as.character(irDesign$asJSON()),
                                           buildOptions = buildOptions)

cat(analysisSql)
#> INSERT INTO myresults.target_def (ref_id, target_cohort_definition_id, target_name)
#> select CAST(1 as int) as ref_id, target_cohort_definition_id, target_name
#> from (
#> select cast(1 as int) as target_cohort_definition_id,
#> cast ('Target cohort 1' as varchar(255)) as target_name
#> ) T
#> ;
#>
#> INSERT INTO myresults.tar_def (ref_id, tar_id, tar_start_with, tar_start_offset, tar_end_with, tar_end_offset)
#> select CAST(1 as int) as ref_id, tar_id, tar_start_with, tar_start_offset, tar_end_with, tar_end_offset
#> FROM (
#> select cast(1 as int) as tar_id,
#> cast ('start' as varchar(10)) as tar_start_with, cast (0 as int) as tar_start_offset,
#> cast ('end' as varchar(10)) as tar_end_with, cast (30 as int) as tar_end_offset
#> ) T
#> ;
#>
#> INSERT INTO myresults.outcome_def (ref_id, outcome_id, outcome_cohort_definition_id, outcome_name, clean_window)
#> select CAST(1 as int) as ref_id, outcome_id, outcome_cohort_definition_id, outcome_name, clean_window
#> from (
#> select cast(1 as int) as outcome_id, cast(2 as int) as outcome_cohort_definition_id,
#> cast ('Outcome 1, 30d Clean' as varchar(255)) as outcome_name,
#> cast (30 as int) as clean_window,
#> cast (0 as int) as excluded_cohort_definition_id
#> ) O
```

```

#> ;
#>
#> INSERT INTO myresults.subgroup_def (ref_id, subgroup_id, subgroup_name)
#> select CAST(1 as int) as ref_id, subgroup_id, subgroup_name
#> FROM (
#> select cast(0 as int) as subgroup_id,
#> cast ('All' as varchar(250)) as subgroup_name
#> UNION ALL
#> select cast(1 as int) as subgroup_id,
#> cast ('Subgroup 1' as varchar(250)) as subgroup_name
#> ) S
#> ;
#>
#>
#> --
#> -- Begin analysis 0
#> --
#>
#> /*****
#> code to implement calculation using the inputs above, no need to modify beyond this point
#>
#> 1) create T + TAR periods
#> 2) create table to store era-fied excluded at-risk periods
#> 3) calculate pre-exclude outcomes and outcomes
#> 4) calculate exclsion time per T/O/TAR/Subject/start_date
#> 5) generate raw result table with T/O/TAR/subject_id, start_date, pe_at_risk (datediff(d,start,end),
#> attach age/gender/year columns
#> 6) Create analysis_ref to produce each T/O/TAR combo
#> 7) perform rollup to calculate IR at the T/O/TAR/S/[age/gender/year] including distinct people and d
#>
#> *****/
#>
#> -- 1) create T + TAR periods
#>
#> --HINT DISTRIBUTE_ON_KEY(subject_id)
#> select subject_id, cohort_definition_id, tar_id, cast(0 as int) as subgroup_id, start_date, end_date
#> into #TTAR_erafied_all
#> FROM (
#> select subject_id, cohort_definition_id, tar_id, min(start_date) as start_date, max(end_date) as e
#>
#> from (
#> select subject_id, cohort_definition_id, tar_id, start_date, end_date, sum(is_start) over (parti
#> from (
#> select subject_id, cohort_definition_id, tar_id, start_date, end_date,
#> case when max(end_date) over (partition by subject_id, cohort_definition_id, tar_id order by
#> from (
#> SELECT COHORT_TAR.subject_id, COHORT_TAR.cohort_definition_id, COHORT_TAR.tar_id, COHORT_TAR
#> case when COHORT_TAR.end_date > op1.observation_period_end_date then op1.observation_perio
#> from (
#> select tc1.cohort_definition_id,
#> tc1.cohort_start_date,
#> tar1.tar_id,

```

```

#>         subject_id,
#>         case
#>             when tar1.tar_start_with = 'start' then DATEADD(day,CAST(tar1.tar_start_offset as int)
#>             when tar1.tar_start_with = 'end' then DATEADD(day,CAST(tar1.tar_start_offset as int),t
#>             else null --shouldnt get here if tar set properly
#>         end as start_date,
#>         case
#>             when tar1.tar_end_with = 'start' then DATEADD(day,CAST(tar1.tar_end_offset as int),tc1
#>             when tar1.tar_end_with = 'end' then DATEADD(day,CAST(tar1.tar_end_offset as int),tc1.c
#>             else null --shouldnt get here if tar set properly
#>         end as end_date
#>     from (
#>         select tar_id, tar_start_with, tar_start_offset, tar_end_with, tar_end_offset
#>         from myresults.tar_def where tar_id in (1) and ref_id = 1
#>     ) tar1,
#>     (
#>         select cohort_definition_id, subject_id, cohort_start_date, cohort_end_date
#>         from demoCohortSchema.cohort
#>         where cohort_definition_id in (1)
#>     ) tc1
#> ) COHORT_TAR
#> inner join mycdm.observation_period op1 on COHORT_TAR.subject_id = op1.person_id
#>     and COHORT_TAR.cohort_start_date >= op1.observation_period_start_date
#>     and COHORT_TAR.cohort_start_date <= op1.observation_period_end_date
#> WHERE COHORT_TAR.start_date <= COHORT_TAR.end_date -- flipped dates are invalid
#>     -- tar must start between same observation period as the cohort_start_date.
#>     and COHORT_TAR.start_date >= op1.observation_period_start_date
#>     and COHORT_TAR.start_date <= op1.observation_period_end_date
#> ) TAR
#> ) ST
#> ) GR
#> GROUP BY subject_id, cohort_definition_id, tar_id, group_idx
#> ) T
#>
#> ;
#>
#>
#> create table #subgroup_person
#> (
#>     subgroup_id bigint NOT NULL,
#>     subject_id bigint NOT NULL,
#>     start_date date NOT NULL
#> );
#>
#> INSERT INTO #subgroup_person (subgroup_id, subject_id, start_date)
#> select distinct cast(1 as int) as subgroup_id, t1.subject_id, t1.start_date
#> FROM #TTAR_erafied_all t1
#> JOIN demoCohortSchema.cohort s1 on t1.subject_id = s1.subject_id
#>     and t1.start_date >= s1.cohort_start_date
#>     and t1.start_date <= s1.cohort_end_date
#> WHERE s1.cohort_definition_id = 300
#> ;
#>

```



```

#> --HINT DISTRIBUTE_ON_KEY(subject_id)
#> select tea.subject_id, tea.cohort_definition_id, tea.tar_id, s.subgroup_id, tea.start_date, tea.end_
#> into #TTAR_erafied_sg
#> FROM #TTAR_erafied_all tea
#> JOIN #subgroup_person s on tea.subject_id = s.subject_id and tea.start_date = s.start_date
#> ;
#>
#> --HINT DISTRIBUTE_ON_KEY(subject_id)
#> SELECT subject_id, cohort_definition_id, tar_id, subgroup_id, start_date, end_date
#> INTO #TTAR_erafied
#> FROM (
#>   SELECT subject_id, cohort_definition_id, tar_id, subgroup_id, start_date, end_date
#>   FROM #TTAR_erafied_all
#>   UNION ALL
#>   SELECT subject_id, cohort_definition_id, tar_id, subgroup_id, start_date, end_date
#>   FROM #TTAR_erafied_sg
#> ) TE;
#>
#> DROP TABLE #TTAR_erafied_all;
#> DROP TABLE #TTAR_erafied_sg;
#>
#> /*
#> 2) create table to store era-fied excluded at-risk periods
#> */
#>
#> --Standard tar exclusion
#> --ways for entry into excluded
#> --1: duration of outcome periods (ex: immortal time due to clean period)
#> --2: other periods excluded (ex: persons post-appendectomy for appendicitis)
#>
#> -- create exclusion eras from cohort clean windows and exclusion cohorts
#> --HINT DISTRIBUTE_ON_KEY(subject_id)
#> select subject_id, outcome_id, min(start_date) as start_date, max(end_date) as end_date
#> into #excluded_tar_cohort
#> from (
#>   select subject_id, outcome_id, start_date, end_date, sum(is_start) over (partition by subject_id,
#>   from (
#>     select subject_id, outcome_id, start_date, end_date,
#>       case when max(end_date) over (partition by subject_id, outcome_id order by start_date rows bet
#>   from (
#>     -- find excluded time from outcome cohorts and exclusion cohorts
#>     -- note, clean window added to event end date
#>     select oc1.subject_id, or1.outcome_id, dateadd(dd,1,oc1.cohort_start_date) as start_date, date
#>     from demoCohortSchema.cohort oc1
#>     inner join (
#>       select outcome_id, outcome_cohort_definition_id, clean_window
#>       from myresults.outcome_def
#>       where outcome_id in (1) and ref_id = 1
#>     ) or1 on oc1.cohort_definition_id = or1.outcome_cohort_definition_id
#>     where dateadd(dd,or1.clean_window, oc1.cohort_end_date) >= dateadd(dd,1,oc1.cohort_start_date)
#>
#>     union all
#>

```

```

#>     SELECT c1.subject_id, or1.outcome_id, c1.cohort_start_date as start_date, c1.cohort_end_date as end_date
#>     FROM demoCohortSchema.cohort c1
#>     inner join (
#>         select outcome_id, excluded_cohort_definition_id
#>         from myresults.outcome_def
#>         where outcome_id in (1) and ref_id = 1
#>     ) or1 on c1.cohort_definition_id = or1.excluded_cohort_definition_id
#>     ) EXCLUDED
#> ) ST
#> ) GR
#> GROUP BY subject_id, outcome_id, group_id;
#>
#> -- intersect the exclusion eras with the time at risk to create the t-tar-outcome exclusion periods
#> --HINT DISTRIBUTE_ON_KEY(subject_id)
#> select  ec1.subject_id,
#>         te1.cohort_definition_id as target_cohort_definition_id,
#>         te1.tar_id,
#>         te1.subgroup_id,
#>         ec1.outcome_id,
#>         case when ec1.start_date > te1.start_date then ec1.start_date else te1.start_date end as start_date,
#>         case when ec1.end_date < te1.end_date then ec1.end_date else te1.end_date end as end_date
#> into #exc_TTAR_o_erafied
#> from #TTAR_erafied te1
#> inner join #excluded_tar_cohort ec1 on te1.subject_id = ec1.subject_id
#>    and ec1.start_date <= te1.end_date
#>    and ec1.end_date >= te1.start_date
#> ;
#>
#> DROP TABLE #excluded_tar_cohort;
#>
#> -- 3) calculate pre-exclude outcomes and outcomes
#> -- calculate pe_outcomes and outcomes by T, TAR, O, Subject, TAR start
#>
#> --HINT DISTRIBUTE_ON_KEY(subject_id)
#> select t1.cohort_definition_id as target_cohort_definition_id,
#>        t1.tar_id,
#>        t1.subgroup_id,
#>        t1.subject_id,
#>        t1.start_date,
#>        o1.outcome_id,
#>        count_big(o1.subject_id) as outcomes_pe,
#>        SUM(case when eo.tar_id is null then 1 else 0 end) as outcomes
#> into #outcome_smry
#> from #TTAR_erafied t1
#> inner join (
#>     select oref.outcome_id, oc.subject_id, oc.cohort_start_date
#>     from demoCohortSchema.cohort oc
#>     JOIN myresults.outcome_def oref on oc.cohort_definition_id = oref.outcome_cohort_definition_id
#>     and oref.ref_id = 1
#>     where oref.outcome_id in (1)
#> ) o1 on t1.subject_id = o1.subject_id
#>    and t1.start_date <= o1.cohort_start_date
#>    and t1.end_date >= o1.cohort_start_date

```

```

#> left join #exc_TTAR_o_erafied eo on t1.cohort_definition_id = eo.target_cohort_definition_id
#> and t1.tar_id = eo.tar_id
#> and t1.subgroup_id = eo.subgroup_id
#> and o1.outcome_id = eo.outcome_id
#> and o1.subject_id = eo.subject_id
#> and eo.start_date <= o1.cohort_start_date
#> and eo.end_date >= o1.cohort_start_date
#> group by t1.cohort_definition_id, t1.tar_id, t1.subgroup_id, t1.subject_id, t1.start_date, o1.outcome_id
#> ;
#>
#> -- 4) calculate exclsion time per T/O/TAR/Subject/start_date
#>
#> --HINT DISTRIBUTE_ON_KEY(subject_id)
#> SELECT EX.target_cohort_definition_id, EX.tar_id, EX.subgroup_id, EX.subject_id, EX.start_date, EX.outcome_id
#> INTO #excluded_person_days
#> FROM (
#> SELECT t1.cohort_definition_id as target_cohort_definition_id,
#> t1.tar_id,
#> t1.subgroup_id,
#> t1.subject_id,
#> t1.start_date,
#> et1.outcome_id,
#> sum(cast((DATEDIFF(day,et1.start_date,et1.end_date) + 1) as bigint)) as person_days
#> FROM #TTAR_erafied t1
#> inner join #exc_TTAR_o_erafied et1 on t1.cohort_definition_id = et1.target_cohort_definition_id
#> and t1.subgroup_id = et1.subgroup_id
#> and t1.tar_id = et1.tar_id
#> and t1.subject_id = et1.subject_id
#> and t1.start_date <= et1.start_date
#> and t1.end_date >= et1.end_date
#> group by t1.cohort_definition_id,
#> t1.subgroup_id,
#> t1.tar_id,
#> t1.subject_id,
#> t1.start_date,
#> et1.outcome_id
#> ) EX;
#>
#> /*
#> 5) aggregate tar and excluded+outcome
#> */
#> WITH tar_overall (target_cohort_definition_id, tar_id, subgroup_id, subject_id, start_date, end_date)
#> AS (
#> SELECT te.cohort_definition_id as target_cohort_definition_id,
#> te.tar_id,
#> te.subgroup_id,
#> te.subject_id,
#> te.start_date,
#> te.end_date,
#> YEAR(te.start_date) - p.year_of_birth as age,
#> p.gender_concept_id as gender_id,
#> YEAR(te.start_date) as start_year
#> FROM #TTAR_erafied te

```

```

#> JOIN mycdm.person p on te.subject_id = p.person_id
#> )
#> select target_cohort_definition_id, tar_id, subgroup_id, age_group_id, gender_id, start_year, person
#> INTO #tar_agg
#> FROM (
#>     SELECT t1.target_cohort_definition_id,
#>         t1.tar_id,
#>         t1.subgroup_id,
#>         cast(null as int) as age_group_id,
#> cast(null as int) as gender_id,
#> cast(null as int) as start_year,
#>     SUM(CAST((DATEDIFF(day,t1.start_date,t1.end_date) + 1) as bigint)) as person_days_pe,
#>     COUNT(distinct t1.subject_id) as persons_at_risk_pe
#> FROM tar_overall t1
#>
#> GROUP BY t1.target_cohort_definition_id, t1.tar_id, t1.subgroup_id
#> ) T_OVERALL
#> ;
#>
#> WITH outcomes_overall (target_cohort_definition_id, tar_id, subgroup_id, outcome_id, subject_id, age
#> AS (
#>     SELECT
#>         t1.cohort_definition_id as target_cohort_definition_id,
#>         t1.tar_id,
#>         t1.subgroup_id,
#>         op.outcome_id,
#>         t1.subject_id,
#>         YEAR(t1.start_date) - p.year_of_birth as age,
#>         p.gender_concept_id as gender_id,
#>         YEAR(t1.start_date) as start_year,
#>         coalesce(e1.person_days, 0) as excluded_days,
#>         DATEDIFF(day,t1.start_date,t1.end_date) + 1 as tar_days,
#>         coalesce(o1.outcomes_pe, 0) as outcomes_pe,
#>         coalesce(o1.outcomes, 0) as outcomes
#> FROM #TTAR_erafied t1
#> JOIN mycdm.person p ON t1.subject_id = p.person_id
#> JOIN ( -- get the list of TTSO of anyone with excluded time or outcomes to limit result
#>     select target_cohort_definition_id, tar_id, subgroup_id, outcome_id, subject_id FROM #excluded_p
#>     UNION -- will remove dupes
#>     select target_cohort_definition_id, tar_id, subgroup_id, outcome_id, subject_id FROM #outcome_sm
#> ) op ON t1.cohort_definition_id = op.target_cohort_definition_id
#>     AND t1.tar_id = op.tar_id
#>     AND t1.subgroup_id = op.subgroup_id
#>     AND t1.subject_id = op.subject_id
#> LEFT JOIN #excluded_person_days e1 ON e1.target_cohort_definition_id = op.target_cohort_definition
#>     AND e1.tar_id = op.tar_id
#>     AND e1.subgroup_id = op.subgroup_id
#>     AND e1.outcome_id = op.outcome_id
#>     AND e1.subject_id = op.subject_id
#>     AND e1.start_date = t1.start_date
#> LEFT JOIN #outcome_smry o1 on o1.target_cohort_definition_id = op.target_cohort_definition_id
#>     AND o1.tar_id = op.tar_id
#>     AND o1.subgroup_id = op.subgroup_id

```

```

#> AND o1.outcome_id = op.outcome_id
#> AND o1.subject_id = op.subject_id
#> AND o1.start_date = t1.start_date
#> )
#> SELECT target_cohort_definition_id, tar_id, subgroup_id, outcome_id, age_group_id, gender_id, start_year,
#> INTO #outcome_agg
#> FROM
#> (
#>     SELECT
#>         t1.target_cohort_definition_id,
#>         t1.tar_id,
#>         t1.subgroup_id,
#>         t1.outcome_id,
#>         cast(null as int) as age_group_id,
#>         cast(null as int) as gender_id,
#>         cast(null as int) as start_year,
#>         SUM(t1.excluded_days) as excluded_days,
#>         -- excluded persons is number of distinct persons minus distinct persons with tar
#>         COUNT(distinct t1.subject_id) - COUNT(distinct case when t1.tar_days > t1.excluded_days then t1.subject_id end) as person_outcomes_pe,
#>         COUNT(distinct case when t1.outcomes_pe > 0 then t1.subject_id end) as person_outcomes,
#>         SUM(t1.outcomes_pe) as outcomes_pe,
#>         SUM(t1.outcomes) as outcomes
#>     FROM outcomes_overall t1
#>
#>     GROUP BY target_cohort_definition_id, tar_id, subgroup_id, outcome_id
#> ) O_OVERALL
#> ;
#>
#> -- 6) Create analysis_ref to produce each T/O/TAR/S combo
#>
#> SELECT t1.target_cohort_definition_id,
#>         tar1.tar_id,
#>         s1.subgroup_id,
#>         o1.outcome_id
#> INTO #tscotar_ref
#> FROM (SELECT target_cohort_definition_id FROM myresults.target_def WHERE target_cohort_definition_id in (1) and ref_id = 1) t1,
#>       (SELECT tar_id FROM myresults.tar_def WHERE tar_id in (1) and ref_id = 1) tar1,
#>       (SELECT subgroup_id FROM myresults.subgroup_def where ref_id = 1) s1,
#>       (SELECT outcome_id FROM myresults.outcome_def WHERE outcome_id in (1) and ref_id = 1) o1
#> ;
#>
#> -- 7) Insert into final table: calculate results via #tar_agg and #outcome_agg for all TSCOTAR combinations
#>
#> INSERT INTO myresults.incidence_summary (ref_id, source_name, target_cohort_definition_id,
#>         tar_id, subgroup_id, outcome_id, age_group_id, gender_id, gender_name, start_year,
#>         persons_at_risk_pe, persons_at_risk, person_days_pe, person_days,
#>         person_outcomes_pe, person_outcomes, outcomes_pe, outcomes,
#>         incidence_proportion_p100p, incidence_rate_p100py)
#> SELECT CAST(1 as int) as ref_id, 'mysource' as source_name, tref.target_cohort_definition_id,
#>         tref.tar_id, tref.subgroup_id, tref.outcome_id, ta.age_group_id, ta.gender_id, c.concept_name as gender_name,
#>         coalesce(ta.persons_at_risk_pe, 0) as persons_at_risk_pe,
#>         coalesce(ta.persons_at_risk_pe, 0) - coalesce(oa.excluded_persons, 0) as persons_at_risk,

```

```

#> coalesce(ta.person_days_pe, 0) as person_days_pe,
#> coalesce(ta.person_days_pe, 0) - coalesce(oa.excluded_days, 0) as person_days,
#> coalesce(oa.person_outcomes_pe, 0) as person_outcomes_pe,
#> coalesce(oa.person_outcomes, 0) as person_outcomes,
#> coalesce(oa.outcomes_pe, 0) as outcomes_pe,
#> coalesce(oa.outcomes, 0) as outcomes,
#> case when coalesce(ta.persons_at_risk_pe, 0) - coalesce(oa.excluded_persons, 0) > 0 then
#>   (100.0 * cast(coalesce(oa.person_outcomes,0) as float) / (cast(coalesce(ta.persons_at_risk_pe, 0) as float)))
#> end as incidence_proportion_p100p,
#> case when coalesce(ta.person_days_pe, 0) - coalesce(oa.excluded_days, 0) > 0 then
#>   (100.0 * cast(coalesce(oa.outcomes,0) as float) / (cast(coalesce(ta.person_days_pe, 0) as float)))
#> end AS incidence_rate_p100py
#> FROM #tscotar_ref tref
#> LEFT JOIN #tar_agg ta ON tref.target_cohort_definition_id = ta.target_cohort_definition_id
#>   AND tref.tar_id = ta.tar_id
#>   AND tref.subgroup_id = ta.subgroup_id
#> LEFT JOIN #outcome_agg oa ON ta.target_cohort_definition_id = oa.target_cohort_definition_id
#>   AND ta.tar_id = oa.tar_id
#>   AND ta.subgroup_id = oa.subgroup_id
#>   AND tref.outcome_id = oa.outcome_id
#>   AND coalesce(ta.age_group_id,-1) = coalesce(oa.age_group_id,-1)
#>   AND coalesce(ta.gender_id,-1) = coalesce(oa.gender_id,-1)
#>   AND coalesce(ta.start_year, -1) = coalesce(oa.start_year,-1)
#> LEFT JOIN mycdm.concept c on c.concept_id = ta.gender_id
#> ;
#>
#> -- CLEANUP TEMP TABLES
#>
#> DROP TABLE #TTAR_erafied;
#> DROP TABLE #subgroup_person;
#> DROP TABLE #exc_TTAR_o_erafied;
#> DROP TABLE #outcome_smry;
#> DROP TABLE #excluded_person_days;
#> DROP TABLE #tscotar_ref;
#> DROP TABLE #tar_agg;
#> DROP TABLE #outcome_agg;
#>
#> --
#> -- End analysis 0
#> --

```

5.2 Render SQL with paramaters and execute

With the previous analysis design and options used to generate the analysisSql, the next step is to render the SQL to provide any remaining parameters, translate, and execute on the database:

```

# if you didn't pass sourceName to buildOptions(), you can render it here
analysisSql <- SqlRender::render(analysisSql, "sourceName" = "OptumDOD")
analysisSql <- SqlRender::translate(analysisSql, "postgresql")

cat(analysisSql)

conn <- DatabaseConnector::connect(connectionDetails)

```

```
DatabaseConnector::executeSql(conn, paste0("DELETE FROM myresults.incidence_summary WHERE ref_id = ", b
DatabaseConnector::executeSql(conn, analysisSql)
DatabaseConnector::disconnect(conn)
```

5.3 Using Temp Tables

Sometimes, it is not convenient or possible to create dedicated tables to store the results. Instead, the `useTempTables` option can be used to place the incidence results into a temp table 'incidence_summary', where they can be ETL'd to another table or exported to a CSV.

The following example demonstrates the additional steps that are necessary if you want to use temp tables:

```
# given the prior irDesign constructed from the previous example
buildOptions <- CohortIncidence::buildOptions(cohortTable = "demoCohortSchema.cohort",
                                              cdmDatabaseSchema = "mycdm",
                                              sourceName = "mysource"
                                              useTempTables = T,
                                              refId = 2)

analysisSql <- CohortIncidence::buildQuery(incidenceDesign = as.character(jsonlite::toJSON(irDesign)),
                                          buildOptions = buildOptions)
analysisSql <- SqlRender::translate(analysisSql, "postgresql")

# if we are using temp tables, the steps to execute the analysis are
# 1) create result temp tables
# 2) execute the analysis query, placing the results into the temp table incidence_summary
# 3) Extract/copy the results from the temp tables
# 4) clean up temp tables

conn <- DatabaseConnector::connect(connectionDetails)

tempDDL <- SqlRender::translate(CohortIncidence::getResultsDdl(useTempTables=T), "postgresql")
DatabaseConnector::executeSql(conn, tempDDL)
DatabaseConnector::executeSql(conn, analysisSql)

# In this example, copy to a permanent table from the temp table, but the results could be downloaded t
exportSql <- SqlRender::translate("insert into mySchema.prefix_incidence_summary select * from #incidence_summary")
DatabaseConnector::executeSql(conn, exportSql)
# or download the results to a dataframe
results <- DatabaseConnector::querySql(conn, SqlRender::translate("select * from #incidence_summary", "j

# use the getCleanupSql to fetch the DROP TABLE expressions for the tables that were created in tempDDL
cleanupSql <- SqlRender::translate(CohortIncidence::getCleanupSql(useTempTables=T), "postgresql")
DatabaseConnector::executeSql(conn, cleanupSql)

DatabaseConnector::dbDisconnect(conn)
```