

Argos: Project Setup & Database Connection

Prerequisites

- 1. Access requirements
 - a. PEDSnet Database
 - b. Bitbucket
 - c. RStudio
- 2. A thorough understanding of how to use Bitbucket (similar to GitHub if you've used that before!). This version control system will ensure there is a centralized location to store your code and make it easier to track changes over time and share with your project collaborators.
 - a. [Here](#) is a documentation for getting started using BitBucket with our Rstudio instance.
- 3. Reviewed our provided [videos and documentation](#) so that you have a general idea of how and why the standard framework is used for queries

Please make sure you have addressed these steps before moving on to the rest of this documentation.

Step 1: Create a new repository & R project

- 1. Log in to RStudio: <https://pedsnetapps.chop.edu/rstudio/>
- 2. Review the [Bitbucket documentation](#) and create a new repository for your study
 - a. If a repository already exists, follow the instructions for the git clone command to pull the repository
- 3. To create a new R project, click File New Project in RStudio
 - a. Follow the prompts to create the project in your existing repository directory
 - b. R projects will set your working directory appropriately each time you open the project and allow you to define project-specific environment variables
 - i. To edit your project-specific environment, run `usethis::edit_r_environ('project')`
 - ii. Global, project-agnostic environment variables can be set with `usethis::edit_r_environ('user')`
- 4. Confirm that you are in the correct project by looking in the top righthand corner of the RStudio interface
- 5. To open your project again, either double click on the .Rproj file in your repository or navigate to your project from the menu in the top right.

Step 2: Set up your database configuration

Before proceeding to the next steps of this section, ensure that you have installed/loaded the [srcr package](#) into your environment. This is what will read the information in your file and turn it into a database connection!

Creating your configuration files

To connect to the database in argos, you will first need to create a `.json` file that contains information about the database you want to connect to and the credentials you would like to use for that connection. Depending on whether you want to connect to Postgres or Trino, there are different sets of information you will want to include in this file.

Anything enclosed in curly brackets is a placeholder that should be replaced with your user / project specific information.

Postgres	Trino
----------	-------

Postgres config

```
{
  "src_name" : "Postgres",
  "src_args" :
  {
    "host" : "my.database.server",
    "port" : 5432,
    "dbname" : "{project_db}",
    "user" : "{username}",
    "password" : "{password}",
    "sslmode" : "require",
    "options" : "-c search_path=dcc_pedsnet,
vocabulary",
    "bigint": "numeric"
  },
  "post_connect_sql": [
    "set role fcc_tier_3;"
  ]
}
```

Trino config

```
{
  "src_name" : "Presto",
  "src_args" : {
    "host" : "{my.database.
server}",
    "port" : 8443,
    "catalog" : "iceberg",
    "schema" : "{schemaname}",
    "user" : "{username}",
    "bigint" : "numeric",
    "use.trino.headers": true
  },
  "pre_connect_fun" : [
    "function (config) {",
    "  library(httr)",
    "  httr::set_config(",
    "    httr::config(httpauth = 1,",
    "      userpwd='{username}:",
    "{password}',",
    "      ssl_verifypeer = 0))",
    "  config",
    "}"
  ],
  "post_connect_sql" : [
    "set role fcc_tier_3"
  ]
}
```

Point to this file in your environment

Once you have created your configuration file, you should create an environment variable that contains the full file path leading to this file. This will ensure both that your file path is not contained within the main body of the code and that argos can reference it to create your connection.

1. Open your project-specific R environment with `usethis::edit_r_environ('project')`
2. Create the an environment variable like so:
 - a. `ARGOS_CONFIG = 'path/to/my/file.json'`
3. Next, open your project specific R profile (`usethis::edit_r_profile('project')`) and add the following snippet, which will help ensure your role (i.e. `fcc_tier_3`) is applied correctly when you are interacting with the data.
 - a. `options('srcr.allow_config_code'=c('sql', 'fun'))`
 - b.  If you do not have 'fun' included here, it may prevent the section of the config with your password from being ingested! If you get an error related to "Unauthorized Access," make sure this is set appropriately
4. Restart your R session to ensure this information is accessible in your environment
5. In the next step, reference the environment variable in the `db_src` parameter while initializing your session



Sometimes, `srcr` may not recognize your file because your path includes the base directory that is automatically applied to the file path during its search. If you get an error saying no valid configuration files could be identified, try removing the beginning of the file path (i.e. `~/`) as a troubleshooting step!

Step 3: Initialize your argos session

Instead of configuring different variables in `run.R` and `site_info.R` as with the old standard framework, these variables have been neatly collapsed into a single function where you can configure your settings all in one place. There are two methods you can use to start your session: one that acts as a more traditional R function (`init_session`) and another that will instead read your variables from a single, unified configuration file (`init_session_from_config`)

To start, open an R file and install / load the argos package

```
pak::pkg_install('PEDSnet/argos')
```

```
library(argos)
```

init_session

init_session operates like a standard R function where you can configure the information for your session in the function parameters. You can start your session with:

```
my_session  argos$new('my_session')$init_session(...)
```

The parameters are documented alongside the function. A few important ones are called out and described below:

Parameter Name	Recommended Setting	Notes
db_src	Sys.getenv('ARGOS_CONFIG')	This parameter is what allows you to connect to the database. You should pass in the file path that points to your JSON configuration file, which should be attached to the environment variable you created in the previous step.
base_dir	Sys.getenv('ARGOS_DATA_REQUEST_ROOT')	argos will automatically look for this environment variable to define your base directory. Usually, this will be the same as the working directory for your project. This tells argos what the top directory is, so it knows where to look for subdirectories like 'specs' or 'results'
results_schema	'my_study_vXX'	When using Trino, we ask that when you create a results_schema on iceberg, please include the database version (i.e. v62, v63) in the name of the schema to help both you and us keep track of which CDM version is being used for the study.
retain_intermediates	TRUE	For Trino configurations, this parameter MUST be set to TRUE. Trino does not have the ability to store temporary tables, so any intermediate tables will need to be retained. This means they will actually appear in your schema when you view it on DBeaver, and you can access them later with results_tbl
cdm	'pedsnet'	This controls the automatic assignment of CDM table name references on the backend, which was originally controlled in the site_info.R file. Here, you can select PEDSnet, OMOP, or PCORnet and argos will know to use the default list of CDM table names associated with the CDM that you pick.
table_case	'lower'	This will change the casing of the default table names set in the prior parameter. The PEDSnet data sources use lowercase, but if you were using a system like Snowflake that typically defaults to uppercase naming, you could change this to "upper." That would allow you to still reference the tables with lowercase in your code, but the system would know to translate it to uppercase when querying the database.
table_names	See note	If you are querying the full CDM, you will not need to configure this parameter as it will be automatically handled as previously described. However, if you are using a mini CDM (where the CDM tables often have prefixes /suffixes), you can use this parameter to set table references like: <pre>list('condition_occurrence' = 'cdm_condition_occurrence_stu123', 'person' = 'cdm_person_stu123', ...)</pre> Taking this approach will allow you to reference the usual table names (i.e. person or condition_occurrence) but instruct the system to query with the mini CDM specific name you provided.
cache_enabled	See note	This parameter defaults to TRUE, which will "cache" any loaded codesets so you don't have to reload them each time you use them for efficiency. However, if you were to make any changes to the codeset, the code would still use the cached value and not reload the new version. So, if you are still in the development phase of a project, we would recommend setting this to FALSE to avoid stale cache codesets while they are undergoing changes. Once your codesets have stabilized, then move this to TRUE to take advantage of the efficiency gains!

init_session_from_config

This function is very similar to init_session, but instead of including all your session information as parameters in the function, you can instead include your database information AND your session information in a single, unified configuration file to create your session. Here is an example of what that would look like when connecting to Trino:

```
my_session  argos$new('my_session')$init_session_from_config(basenames = 'argos_init_session_config',  
                                                             dirs = {path to directory where file lives})
```

argos_init_session_config

```
{
  "argos_config" :
  {
    "src_name" : "Presto",
    "src_args" : {
      "host" : "{my.database.server}",
      "port" : 8443,
      "catalog" : "iceberg",
      "schema" : "{schemaname}",
      "user" : "{username}",
      "bigint" : "numeric",
      "use.trino.headers": true
    },
    "pre_connect_fun" : [
      "function (config) {",
      "  library(httr)",
      "  httr::set_config(",
      "    httr::config(httpauth = 1,",
      "      userpwd='{username}:{password}',",
      "      ssl_verifypeer = 0))",
      "  config",
      "}"
    ],
    "post_connect_sql" : [
      "set role fcc_tier_3"
    ]
  },
  "cdm_schema" : "{cdm_schema_vXX}",
  "vocabulary_schema" : "{vXX_vocabulary}",
  "results_schema" : "{my_schema_vXX}",
  "cdm" : "pedsnet",
  "table_case" : "lower",
  "retain_intermediates" : true,
  "cache_enabled" : false
}
```

Note that your full database connection information should be included in the **argos_config** section of this file, and your other session parameters should be listed outside of that section. If you were connecting to Postgres, you would just replace the information in **argos_config** with your Postgres config information (and leave the other parameters the same). Just like with other R functions, as long as the parameter has a default value, you do not need to include it here. You will need to at least provide your database connection information and the schemas (cdm_schema, vocabulary_schema, results_schema), and for Trino, we recommend configuring retain_intermediates.

RPresto code patch

In order to use argos to access Trino, your repository should contain the following code snippets to apply a small patch to the RPresto package (which facilitates connection to Trino via R). We recommend placing these in the same file where you initialize your session so they are applied consistently each time you connect to Trino.

This patch will ensure that compute_new, copy_to_new, and output_tbl work appropriately. This is a temporary solution while we wait for the issue with the underlying package to be addressed (issue described [here](#)).

RPresto patch

```
.sqlCreateTableAs <- function(con, name, sql, with = NULL, ...) {
  name <- DBI::dbQuoteIdentifier(con, name)
  DBI::SQL(paste0(
    "CREATE TABLE ", name, "\n",
    if (!is.null(with)) paste0(with, "\n"),
    "AS\n",
    sql
  ))
}

setMethod("sqlCreateTableAs", signature("PrestoConnection"), .sqlCreateTableAs)

assignInNamespace(
  ".compute_tbl_presto",
  function(x, name, temporary = FALSE, ..., cte = FALSE) {
    if (rlang::is_bare_character(x) || dbplyr::is.ident(x) || dbplyr::is.sql(x)) {
      name <- unname(name)
    }
    if (identical(cte, TRUE)) {
      if (inherits(x$lazy_query, "lazy_base_remote_query")) {
        stop(
          "No operations need to be computed. Aborting compute.",
          call. = FALSE
        )
      }
      con <- dbplyr::remote_con(x)
      # We need to specify sql_options here so that use_presto_cte is passed to
      # db_sql_render correctly
      # (see https://github.com/tidyverse/dbplyr/issues/1394)
      sql <- dbplyr::db_sql_render(
        con = dbplyr::remote_con(x), sql = x,
        sql_options = dbplyr::sql_options(), use_presto_cte = FALSE
      )
      con@session$addCTE(name, sql, replace = TRUE)
    } else {
      sql <- dbplyr::db_sql_render(
        dbplyr::remote_con(x), x, use_presto_cte = TRUE
      )
      name <- dbplyr::db_compute(
        dbplyr::remote_con(x), name, sql, temporary = temporary, ...
      )
    }
    name
  }, ns='RPresto'
)
```

Step 4: Set your default session & start analyzing!

Once you have created your session, you should set it as your default so argos knows which set of information to use for downstream functions:

```
set_argos_default(my_session)
```

Now you are ready to start your project!!

Trino-specific computation needs

If you are using Trino, there are just a few small differences in the SQL dialect you will have to keep in mind when conducting your analysis in R:

--	--

Date computations	Presto SQL has its own date computation functions that can be embedded directly into your R code when working with remote database tables. Their documentation is very helpful and we recommend reading about it here .
String searching & manipulation	grep / grepl and most stringr functions are not compatible with the Trino backend. They have their own string & regular expression based functions that should be embedded into your code instead.
Large local tables	When you have a table that you have collected locally to your R environment and would like to write out to the database, sometimes this generates a very large query that exceeds the limit we have set. In order to still output your table, there is a new parameter to output_tbl that will allow you to output your table in smaller chunks. You will need to iterate on the chunk size to determine what is best for your table – there is no one size fits all! <pre>output_tbl(my_tbl, 'my_tbl_name', .chunk_size = 5000)</pre>

Additional Documentation

- Presto SQL functions: <https://prestodb.io/docs/current/functions.html>
- Recording of argos demonstration & associated sample code: [Data Analytics - August 14, 2026.mp4](#)