In dbt cloud the schema are defined in the credentials of the project.

But if we want to select the separate schema for each project then it can be defined in the dbt\_project.yml file

models:

  my\_new\_project:

    # Applies to all files under models/example/

    example:

      materialized: table

      chicagocrime:

        schema: chic\_crime # optional because in dbt cloud it is already defined but in settings of project - after defining schema again here it will combine it with the one defined in settings

        tags:

          - crime

      raw\_zone:

        schema: raw\_zone

        tags:

          - healthcare

          - health

So now the tables that would be created will add the previous schema name with the new and create a whole new schema to store tables in it

* Like the chicagocrimedata will be stored in the schema named as **star\_schema\_chic\_crime**
* The healthcare data will be stored in the schema named as **star\_schema\_raw\_zone**
* *This duplication of the name is just because that in the backend it is defined in the way that dbt target schema + new schema = custom schema so the target schema is to be removed and again this kind of macro code is to be placed in macros and it will solve the problem*

When first using custom schemas, it's common to assume that a model will be built in a schema that matches the schema configuration exactly, for example, a model that has the configuration schema: marketing, would be built in the marketing schema. However, dbt instead creates it in a schema like <target\_schema>\_marketing by default – there's a good reason for this!

In a typical setup of dbt, each dbt user will use a separate target schema (see Managing Environments). If dbt created models in a schema that matches a model's custom schema exactly, every dbt user would create models in the same schema.

Further, the schema that your development models are built in would be the same schema that your production models are built in! Instead, concatenating the custom schema to the target schema helps create distinct schema names, reducing naming conflicts. If you prefer to use different logic for generating a schema name, you can change the way dbt generates a schema name

Custom schema

We can configure the schemas in the models too while creating it

{{ config(tags=["crime"], schema = "chic") }}

with chicagocrime as (select \* from {{ source('src\_chicagocrime', 'chicagocrimedata') }})

select \*

from chicagocrime

we can add schema to the tests separately and combinely

separately to each test

version: 2

models:

  - name: dim\_location

    description: Check whether the values of fact\_id is unique or not

    columns:

      - name: location\_id

        tags: ['loc\_tag']

        tests:

          - unique:

              store\_failures: true

              schema: "loc\_test\_uniq"

          - not\_null

Combine in the dbt\_project file

# using the `{{ config(...) }}` macro.

  models:

    my\_new\_project:

      # Applies to all files under models/example/

      example:

        materialized: table

        chicagocrime:

          chicagocrimedata:

          schema: chic\_crime

          tags:

            - crime

          tests:

            config:

              +store\_failures: false

        raw\_zone:

          schema: raw\_zone

          tags:

            - healthcare

            - health

Target\_schema\_custom\_schema

So as we have faced this problem that when we add up a custom schema it is concatenated with target schema

***This Is good for a reason that the duplication will not occur and it will give unique names***

But if we want to save our data in the custom schema that we had defined then there is a predefined macro named as generate\_name\_schema

Working:

We will have to copy that macro and create a new file in macro folder and paste that code

**{% macro generate\_schema\_name(custom\_schema\_name, node) -%}**

**{%- set default\_schema = target.schema -%}**

**{%- if custom\_schema\_name is none -%}**

**{{ default\_schema }}**

**{%- else -%}**

**{{ default\_schema }}\_{{ custom\_schema\_name | trim }}**

**{%- endif -%}**

**{%- endmacro %}**

Now we will be having some changes in this code to get our required results

**{% macro generate\_schema\_name(custom\_schema\_name, node) -%}**

**{%- set default\_schema = target.schema -%}**

**{%- if custom\_schema\_name is none -%}**

**{{ default\_schema }}**

**{%- else -%}**

**{{ custom\_schema\_name | trim }}**

**{%- endif -%}**

**{%- endmacro %}**

***If your dbt project has a macro that’s also named generate\_schema\_name, dbt will always use the macro in your dbt project instead of the default macro.***

**The above method is not correct because**

❗️ Warning: Don't replace default\_schema in the macro.

If you're modifying how dbt generates schema names, don't just replace {{ default\_schema }}\_{{ custom\_schema\_name | trim }} with {{ custom\_schema\_name | trim }} in the generate\_schema\_name macro.

Removing {{ default\_schema }} causes developers to overriding each other's models when custom schemas are defined. This can also cause issues during development and continuous integration (CI).

you'll need **a generate\_schema\_name** macro in your project that points to this logic. You can do this by creating a file in your macros directory (typically named **get\_custom\_schema.sql),** and copying/pasting the following code:

**macros/get\_custom\_schema.sql**

*-- put this in* ***macros/get\_custom\_schema.sql***

**{% macro generate\_schema\_name(custom\_schema\_name, node) -%}**

**{{ generate\_schema\_name\_for\_env(custom\_schema\_name, node) }}**

**{%- endmacro %}**

**Note:** When using this macro, you'll need to set the target name in your job specifically to "prod" if you want custom schemas to be applied.

**A data analyst wants to specify a custom list of directories where singular tests are located. Which of the following options allows them to achieve this in dbt\_project.yml file?**

Choose only ONE best answer.

**A**

test-paths: ["tests"]

**B**

test-paths: ["/path/to/custom\_tests"]

**C**

test-paths: [directorypath]

**D**

test-paths: ["custom\_tests"]

**This answer is incorrect. The correct answer is 'D'**

Answer: D) test-paths: ["custom\_tests"]

Explanation: The dbt\_project.yml file allows data analysts to optionally specify a custom list of directories where singular tests are located.

The default configuration is "test-paths: ['tests']", where dbt will search for tests in the tests directory. However, we can use a subdirectory named custom\_tests instead of tests for data tests, they can use the following configuration: "test-paths: ['custom\_tests']".

Option A is the default configuration, which specifies that dbt will search for tests in the tests directory.

Option B specifies an absolute path to the custom\_tests directory, which may not be necessary and can cause issues if the path changes.

 Option C is not a valid option and is not mentioned in the given text. Test-paths can be tested by creating test files in the specified directories and running them with the "dbt test" command. If the specified directories contain test files, dbt will use them instead of the default "tests" directory. If there are no test files in the specified directories, dbt will fall back to the default configuration.

**How can you disable query comments in dbt\_project.yml?**

Choose only ONE best answer.

**A**

Set query-comment: "disable"

**B**

Set query-comment: "none"

**C**

Leave query-comment blank or set query-comment: null

**D**

Set query-comment: "no\_comment"