Graph Operators:

The "plus" operator[​](https://docs.getdbt.com/reference/node-selection/graph-operators#the-plus-operator)

If placed at the front of the model selector, + will select all parents of the selected model. If placed at the end of the string, + will select all children of the selected model.

**$ dbt run --select my\_model+** *# select my\_model and all children*  
**$ dbt run --select +my\_model** *# select my\_model and all parents*  
**$ dbt run --select +my\_model+** *# select my\_model, and all of its parents and children*

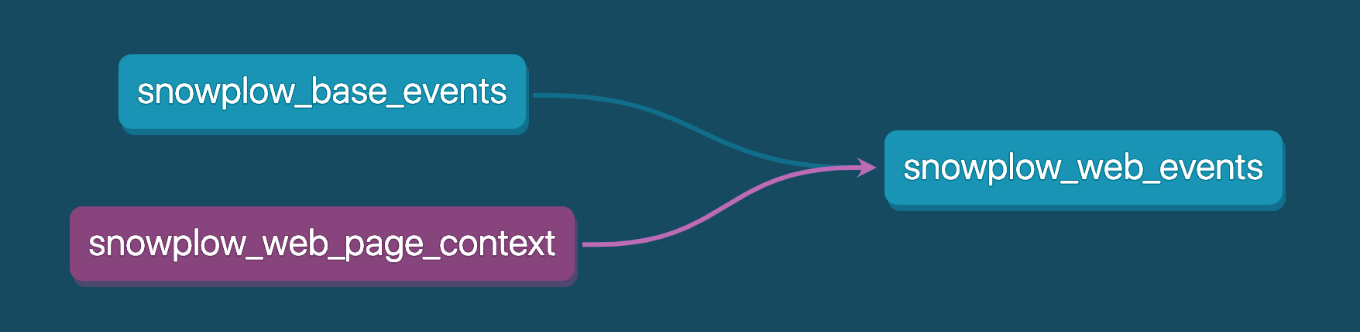
The "n-plus" operator[​](https://docs.getdbt.com/reference/node-selection/graph-operators#the-n-plus-operator)

You can adjust the behavior of the + operator by quantifying the number of edges to step through.

**$ dbt run --select my\_model+1**  *# select my\_model and its first-degree children*  
**$ dbt run --select 2+my\_model** *# select my\_model, its first-degree parents, and its second-degree parents ("grandparents")*  
**$ dbt run --select 3+my\_model+4** *# select my\_model, its parents up to the 3rd degree, and its children down to the 4th degree*

The "at" operator[​](https://docs.getdbt.com/reference/node-selection/graph-operators#the-at-operator)

The @ operator is similar to +, but will also include *the parents of the children of the selected model*. This is useful in continuous integration environments where you want to build a model and all of its children, but the *parents* of those children might not exist in the database yet. The selector **@snowplow\_web\_page\_context** will build all three models shown in the diagram below.

[](https://docs.getdbt.com/reference/node-selection/graph-operators)@snowplow\_web\_page\_context will select all of the models shown here

**$ dbt run --models @my\_model** *# select my\_model, its children, and the parents of its children*

The "star" operator[​](https://docs.getdbt.com/reference/node-selection/graph-operators#the-star-operator)

The \* operator matches all models within a package or directory.

**$ dbt run --select snowplow.\***  *# run all of the models in the snowplow package*  
**$ dbt run --select finance.base.\*** *# run all of the models in models/finance/base*

Set Operators - imp

Unions[​](https://docs.getdbt.com/reference/node-selection/set-operators#unions)

Providing multiple space-delineated arguments to the --select or --exclude flags selects the union of them all. If a resource is included in at least one selector, it will be included in the final set.

* **Run snowplow\_sessions, all ancestors of snowplow\_sessions, fct\_orders, and all ancestors of fct\_orders:**

$ dbt run --select +snowplow\_sessions +fct\_orders

***(If we are targeting the union among two or three models or folders than we will write these with a space and the operators with it. Space between the models or folders when there names are written in the selector with operators )***

Intersections[​](https://docs.getdbt.com/reference/node-selection/set-operators#intersections)

If you separate multiple arguments for --select and --exclude with commas and no whitespace in between, dbt will select only resources that satisfy *all* arguments.

1. **Run all the common ancestors of snowplow\_sessions and fct\_orders:**

$ dbt run --select +snowplow\_sessions,+fct\_orders

1. **Run all the common descendents of stg\_invoices and stg\_accounts:**

$ dbt run --select stg\_invoices+,stg\_accounts+

1. **Run models that are in the marts/finance subdirectory *and* tagged nightly:**

$ dbt run --select marts.finance,tag:nightly

(***for intersection or to select the common value, mode or file among them. We will put a comma between the models when selected and then it will take out most common values between the selected models or files)***

Conclusion

When you want to print or run the resource which may be location in one of the models or files then we will be using the union operator because it will combine the selected models and find out the required resource amoung them. We will have to mention the model names with only a single space between them

$ dbt run --select +snowplow\_sessions +fct\_orders

When you want to run the a common resource or file between multiple models or files we will have to put them in the run command with selector with comma between them. By running this command it will scan and give the most common resources in both of these models or files

$ dbt run --select +snowplow\_sessions,+fct\_orders

Exclude

**Excluding models**[**​**](https://docs.getdbt.com/reference/node-selection/exclude#excluding-models)

dbt provides an --exclude flag with the same semantics as --select. Models specified with the --exclude flag will be removed from the set of models selected with --select.

*# select all models in my\_package and their children except a\_big\_model and its children*

**$ dbt run --select my\_package.\*+ --exclude my\_package.a\_big\_model+**

**Exclude a specific resource by its name or lineage:**

*# test*  
$ dbt test --exclude not\_null\_orders\_order\_id

*# test all models except the not\_null\_orders\_order\_id test*  
$ dbt test --exclude orders

*# test all models except tests associated with the orders model*  
  
*# seed*  
$ dbt seed --exclude account\_parent\_mappings

*# load all seeds except account\_parent\_mappings*  
  
*# snapshot*  
$ dbt snapshot --exclude snap\_order\_statuses

*# execute all snapshots except snap\_order\_statuses*

**Methods**:

***Selector methods return all resources that share a common property, using the syntax method:value.***

While it is recommended to explicitly denote the method, you can omit it (the default value will be one of path, file or fqn) .

***When using methods, it will possess the property of intersection. Like the operation will be applied on the common things found in the resources or methods***

1. The "tag" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-tag-method)

The **tag:** method is used to select models that match a specified [tag](https://docs.getdbt.com/reference/resource-configs/tags).

$ dbt run --select tag:nightly *# run all models with the `nightly` tag*  
***(now dbt will run only those models having tag nightly)***

1. The "source" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-source-method)

The source method is used to select models that select from a specified [source](https://docs.getdbt.com/docs/build/sources#using-sources). Use in conjunction with the + operator.

$ dbt run --select source:snowplow+ *# run all models that select from Snowplow sources*  
***(Run all the models that have the same source which is snowplow. )***

1. The "path" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-path-method)

***The path method is used to select models/sources defined at or under a specific path.*** Model definitions are in SQL/Python files (not YAML), and source definitions are in YAML files. While the path prefix is not explicitly required, ***it may be used to make selectors unambiguous.***

*# These two selectors are equivalent*  
dbt run --select path:models/staging/github  
dbt run --select models/staging/github  
  
*# These two selectors are equivalent*  
dbt run --select path:models/staging/github/stg\_issues.sql  
dbt run --select models/staging/github/stg\_issues.sql

1. The "file" or "fqn" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-file-or-fqn-method)

***The file or fqn method can be used to select a model by its filename, including the file extension (.sql).***

*# These are equivalent*  
dbt run --select some\_model.sql  
dbt run --select some\_model  
dbt run --select fqn:some\_model *# fqn is an abbreviation for "****fully qualified name"***

1. The "package" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-package-method)

The package method is used to select models defined within the root project or an installed dbt package***. While the package: prefix is not explicitly required, it may be used to make selectors unambiguous.***

***The "package" selector targets the entire package and its contents, but it does not implicitly select models or tests based on the packages they use.***

*# These three selectors are equivalent*  
dbt run --select package:snowplow  
dbt run --select snowplow  
dbt run --select snowplow.\*

1. The "config" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-config-method)

The config method is used to select models that match a specified [node config](https://docs.getdbt.com/reference/configs-and-properties).

***This method is used to select the models or tests on the basis of their configurations i.e., materialization,schema,model,cluster,keys,tags,grants etc***

$ dbt run --select config.materialized:incremental *# run all models that are materialized incrementally*  
$ dbt run --select config.schema:audit *# run all models that are created in the `audit` schema*  
$ dbt run --select config.cluster\_by:geo\_country *# run all models clustered by `geo\_country`*

**While most config values are strings, you can also use the config method to match boolean configs, dictionary keys, and values in lists.**

For example, given a model with the following configurations:

*{{ config(  
 materialized = 'incremental',  
 unique\_key = ['column\_a', 'column\_b'],  
 grants = {'select': ['reporter', 'analysts']},  
 transient = true  
) }}*  
  
select ...

*You can select using any of the following:*

$ dbt ls -s config.materialized:incremental  
$ dbt ls -s config.unique\_key:column\_a  
$ dbt ls -s config.grants.select:reporter  
$ dbt ls -s config.transient:true

***These commands are used to list all the resources on the basis of their config settings!!***

1. The "test\_type" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-test_type-method)

The test\_type method is used to select tests based on their type, singular or generic:

$ dbt test --select test\_type:generic *# run all generic tests*  
$ dbt test --select test\_type:singular *# run all singular tests*

**The "test\_name" method**[**​**](https://docs.getdbt.com/reference/node-selection/methods#the-test_name-method)

The test\_name method is used to select tests based on the name of the generic test that defines it. For more information about how generic tests are defined, read about [tests](https://docs.getdbt.com/docs/build/tests).

$ dbt test --select test\_name:unique *# run all instances of the `unique` test*  
$ dbt test --select test\_name:equality *# run all instances of the `dbt\_utils.equality` test*  
$ dbt test --select test\_name:range\_min\_max *# run all instances of a custom schema test defined in the local project, `range\_min\_max`*

In dbt, the **"test name"** method refers to a selector that allows you to target specific tests based on their names. It enables you to run or perform actions on specific tests within your dbt project.

When using the **"test name"** method, *you provide the name or pattern of the test you want to select. Dbt will identify and execute the tests that match the specified name or pattern.*

Here's an example to illustrate the usage of the **"test name"** method:

Let's say you have several tests in your dbt project, including **"test\_sales\_totals",** "**test\_customer\_counts**", and **"test\_inventory\_validation**".

To select and run the **"test\_sales\_totals"** test specifically, you can use the **"test name"** method like this:

dbt test --select test\_name:test\_sales\_totals

This command will execute only the **"test\_sales\_totals"** test, allowing you to focus on testing that specific aspect of your data.

*The* ***"test name****" method supports patterns as well, using wildcards or regular expressions*.

For example, to select and run all tests with names starting with **"test\_customer",** you can use the following command:

dbt test --select test\_name:test\_customer\*

This will execute tests like **"test\_customer\_counts"** and **"test\_customer\_segmentation".**

Using the **"test name"** method, you can run or perform actions on specific tests within your dbt project, allowing you to focus on testing specific aspects of your data models or validating particular business logic.

1. The "state" method

The **"state"** method in dbt helps you select and work with specific parts of your project based on their changes compared to a previous version. It's like a tool that lets you focus on the new or modified parts.

***Imagine you have a project with different elements like models, tests, and sources. When you make changes to these elements, dbt can compare them to the previous version using a file called the manifest. The manifest keeps track of the project's structure and state.***

*With the* ***"state" method,*** *you can select specific elements based on their state compared to the previous version.*

For example:

* state:new: Selects elements that are completely new and not in the previous version. There is no node with the same unique\_id in the comparison manifest
* state:modified: Selects elements that are new or have been changed from the previous version. All new nodes, plus any changes to existing nodes

By using different dbt commands with the "state" method, you can perform actions on these selected elements. For example:

* dbt test --select state:new: Runs tests on new models and new tests on old models.
* dbt run --select state:modified: Runs models that have been modified.
* dbt ls --select state:modified: Lists all modified elements (not just models).

The "state:modified" criterion can be further refined using subselectors to specify specific types of changes, such as changes to the content of the elements or their configurations.

The "state" method is a powerful way to work with parts of your project based on their changes compared to the previous version. It helps you focus on what's new or modified, making it easier to manage and track changes in your dbt project.

1. The "exposure" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-exposure-method)

The exposure method is used to select parent resources of a specified [exposure](https://docs.getdbt.com/docs/build/exposures). Use in conjunction with the + operator.

**Exposure is basically used to expose the dependency of models or sources on each other and the relation or details among them**

$ dbt run --select +exposure:weekly\_kpis *# run all models that feed into the weekly\_kpis exposure*

***This command is used to run all the models that are in the exposure that are parents and exposure is made up of and in upstreams or parents models it will go with*** weekly\_kpis like run those exposures having weekly\_kpis in the name of exposures

$ dbt test --select +exposure:\* *# test all resources upstream of all exposures*

***In this command there is no limitations, it will run all the exposures in the upstreams or parents fed in the exposures.***  
$ dbt ls --select +exposure:\* --resource-type source *# list all sources upstream of all exposures*  
**it will create a list of all the sources listed in the exposure and are upstream or parents**

1. The "metric" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-metric-method)

The metric method is used to select parent resources of a specified [metric](https://docs.getdbt.com/docs/build/metrics). Use in conjunction with the + operator.

$ dbt build --select +metric:weekly\_active\_users *# build all resources upstream of* ***weekly\_active\_users*** *metric*  
$ dbt ls --select +metric:\* --resource-type source *# list all source tables upstream of all metrics*

1. The "result" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-result-method)

The result method is related to the state method described above and can be used to select resources based on their result status from a prior run. Note that one of the dbt commands [run, test, build, seed] must have been performed in order to create the result on which a result selector operates. You can use result selectors in conjunction with the + operator.

$ dbt run --select result:error --state path/to/artifacts *# run all models that generated errors on the prior invocation of dbt run*  
$ dbt test --select result:fail --state path/to/artifacts *# run all tests that failed on the prior invocation of dbt test*  
$ dbt build --select 1+result:fail --state path/to/artifacts *# run all the models associated with failed tests from the prior invocation of dbt build*  
$ dbt seed --select result:error --state path/to/artifacts *# run all seeds that generated errors on the prior invocation of dbt seed.*

When you run dbt commands such as **dbt run**, **dbt test**, **dbt build**, or **dbt seed**, dbt keeps track of the result status of each executed resource (models, tests, seeds) during that run. The result status indicates whether the resource was successful, had errors, failed tests, or other relevant outcomes.

The "result" method allows you to select resources based on their result status from a previous run of dbt. For example, if you run **dbt run** and encounter errors in some models, you can use the "result" method to select and perform actions on those specific models that generated errors.

Here's an example to illustrate the usage:

Suppose you ran **dbt run** and encountered errors in some models. The following command will select and run only the models that generated errors in the previous dbt run:

dbt run --select result:error --state path/to/artifacts

In this case, the "result:error" selector specifies that you want to target resources (models) that had errors as their result status. By providing the path to the artifacts or manifest file via the **--state** flag, dbt will use the result information from the previous run and execute the specified models that had errors.

Similarly, you can use the "result" method with other dbt commands (**test**, **build**, **seed**) and different result statuses to select and perform actions on specific resources based on their prior results.

The purpose of using the "result" method is to selectively rerun or perform actions on resources that had specific outcomes in a previous dbt run, allowing you to focus on troubleshooting or addressing the resources that need attention based on their result status.

1. The "source\_status" method[​](https://docs.getdbt.com/reference/node-selection/methods#the-source_status-method)

Another element of job state is the source\_status of a prior dbt invocation. After executing dbt source freshness, for example, dbt creates the sources.json artifact which contains execution times and max\_loaded\_at dates for dbt sources.

***The following dbt commands produce sources.json artifacts whose results can be referenced in subsequent dbt invocations:***

* dbt source freshness

After issuing one of the above commands, you can reference the source freshness results by adding a selector to a subsequent command as follows:

*# You can also set the DBT\_ARTIFACT\_STATE\_PATH environment variable instead of the --state flag.*  
$ dbt source freshness *# must be run again to compare current to previous state*  
$ dbt build --select source\_status:fresher+ --state path/to/prod/artifacts

Conclusion

State is the method which is used to compare the older version and latest or updated version of the model or test or source or seeds.

In results the previous tests are recorded and the record is present what was the outcome of the result (pass,failed,error etc) so the results are used to create a reference to those which have failed. And we use state method because we are using the previous version and the paths are given to the artifacts (logs of resources being run) and those models are run, test, source, seed which have failed previously. The failed logs are in the artifacts and with state method the previous version is used and result method is used to make reference to those which have been failed.