Generating Specifications

To generate specifications, ensure you've provided the necessary details in the config.json file.

Write Config.json

Here's a breakdown of the specifications required within config.json:

1. set_op_types: This field encompasses various types such as "UNION", "UNION ALL", "INTERSECT", "EXCEPT", "MINUS", and "none". Specify both first_query and second_query. If you designate none as the set_op_types, it will disregard the second_query.

```
Unset
{"set_op_types":["UNION", "UNION ALL", "INTERSECT", "EXCEPT","MINUS","none"],
"first_query": {...},
"second_query": {...}}
```

To create a query specification, select the parameters listed below:

2. join_types: For self-joins, specify whether you intend to use INNER JOIN for two or more identical tables, assigning them alias names."

```
Unset
  "join_types":["INNER JOIN", "LEFT JOIN", "RIGHT JOIN", "FULL OUTER JOIN", "SELF
JOIN"],
```

3. table_exp_types: If you specify join_3, it will randomly select three join_types to create a FROM clause. For instance, the types could be: "INNER JOIN", "LEFT JOIN", "SELF JOIN"

```
Unset
 "table_exp_types":["single_table",
    "single_table_with_name_changing",
    "join_1",
    "join_2",
    "join_3"]
```

4. where_clause_types:

```
Unset
"where_clause_types" : [
  "none",
  "between",
  "in_set",
  "pattern_matching",
  "null_check",
  "basic_comparison",
  "logical_operators",
  "subquery"],
```

5. subquery_in_where: if you specify subqury in where_clause_types you need to specify this parameter.

```
Unset
 "subquery_in_where":[
      "in_with_subquery",
      "not_in_with_subquery"
      "comparison_with_subquery",
      "not_exists_subquery",
      "exists_subquery"],
```

6. min_max_depth_in_subquery:

```
Unset
"min_max_depth_in_subquery":[3,5],
```

7. You need to specify:
    a. in_set when in_set is in where_clause_types,
    b. like_or_not_like and pattern_matching_types when pattern_matching is in where_clause_types
    c. basic_comp_ops when basic_comparison in where_clause_types
    d. null_operators when null_check in where_clause_types

```
Unset
 "in_set":["IN", "NOT IN"],
"like_or_not_like" :["LIKE", "NOT LIKE"],
"basic_comp_ops": ["=", "<>", "!=", ">", "<", ">=", "<="],
"pattern_matching_types": [
```

```
  "starts_with_a",
  "ends_with_ing",
  "exactly_5_characters",
  "does_not_contain_xyz"
],
"null_operators" : ["IS NULL", "IS NOT NULL"],
```

8. meaningful_joins: if join_# in table_exp_types you need to specify this.

```
Unset
"meaningful_joins":["yes", "no","mixed"],
```

9. You need to specify:
    a. having_types when number_of_valu_exps_in_group_by != [0]
    b. aggregate_functions_for_having when number_of_valu_exps_in_group_by != [0],
    c. subquery_in_having when number_of_valu_exps_in_group_by != [0] and “subquery” is in having_types

```
Unset
 "number_of_value_exps_in_group_by":[0,1,2,3,4,5],
"having_types" : [
    "none",
    "single",
    "multiple",
    "subquery"
],
"aggregate_functions_for_having":[
"MAX", "MIN", "AVG", "SUM", "COUNT", "COUNT DISTINCT"],
"subquery_in_having":["in_with_subquery",
"comparison_with_subquery","not_exists_subquery","exists_subquery"],
```

10. You need to specify:
    a. string_func_col if string_func_exp is in value_exp_types
    b. agg_col if agg_exp is in value_exp_types
    c. arithmatic_col if arithmatic_exp is in value_exp_types
    d. count_distinct_col if count_distinct_exp is in value_exp_types
        i. These attributes are used to specify that we want to get alias name for that value expression type or not.

```
Unset

"number_of_value_exps_in_select":[1,2,"*"],
"value_exp_types": [
    "single_exp",
    "alias_exp",
    "arithmatic_exp",
    "string_func_exp",
    "agg_exp",
    "count_distinct_exp",
    "subquery_exp"],
"string_func_col":["alias", "no_alias"],
"agg_col":["alias"],
"arithmatic_col":["alias", "no_alias"],
"count_distinct_col":["alias", "no_alias"],
"distinct_types": ["none", "distinct"],
```

11. Min_max_depth_in_subquery is optional attribute. The default can be set to [0,0]

```
Unset
  min_max_depth_in_subquery = specs.get("min_max_depth_in_subquery", [0, 0])
```

12. join_types: for self join you can specify that you want to INNER JOIN 2 or more same tables with alias names.

```
Unset
"orderby_types": ["ASC", "DESC","number_ASC","number_DESC" ,"none","multiple"],
"limit_types" : ["none", "without_offset", "with_offset"]
```

Here is the examples of specifications generated:

```
Unset
"farm": {
    "cd0d026f3b332f175e976d844d6fef1bbb724985": {
      "set_op_type": "none",
      "first_query": {
        "meaningful_joins": "no",
        "table_exp_type": "FULL OUTER JOIN_INNER JOIN_SELF JOIN",
        "where_type": {
```

```
        "logical_operator": [
          "OR",
          "pattern_matching",
          "comparison_with_subquery"
        ]
      },
      "number_of_value_exp_in_group_by": 3,
      "having_type": {
        "single": "MIN"
      },
      "orderby_type": "none",
      "limit_type": "without_offset",
      "value_exp_types": [
        "single_exp_text",
        "string_func_exp"
      ],
      "distinct_type": "none",
      "min_max_depth_in_subquery": [
        3,
        5
      ]
    }
  },
```

If we have set_op:

Unset

```
"farm": {
    "4477535e2050b7998ba56d8bd35b558ccb165a91": {
      "set_op_type": "INTERSECT",
      "first_query": {
        "meaningful_joins": "no",
        "table_exp_type": "INNER JOIN_INNER JOIN_RIGHT JOIN",
        "where_type": {
          "logical_operator": [
            "OR",
            "basic_comparison",
            "not_exists_subquery"
          ]
        },
        "number_of_value_exp_in_group_by": 0,
        "having_type": "none",
```

```
      "orderby_type": "number_ASC",
      "limit_type": "none",
      "value_exp_types": [
        "arithmatic_exp_alias",
        "count_distinct_exp"
      ],
      "distinct_type": "distinct",
      "min_max_depth_in_subquery": [
        0,
        0
      ]
    },
    "second_query": {
      "meaningful_joins": "mixed",
      "table_exp_type": "INNER JOIN_LEFT JOIN_RIGHT JOIN",
      "where_type": {
        "logical_operator": [
          "OR",
          "IN",
          "pattern_matching"
        ]
      },
      "number_of_value_exp_in_group_by": 0,
      "having_type": "none",
      "orderby_type": "none",
      "limit_type": "none",
      "value_exp_types": [
        "arithmatic_exp_alias",
        "count_distinct_exp"
      ],
      "distinct_type": "distinct",
      "min_max_depth_in_subquery": [
        0,
        0
      ]
    }
  },
```

Randomlly choose from all possible combinations

- config_file stores the path to "config_file.json" if you create another config file need to change this path
    - config_file = os.path.abspath(os.path.join(current_dir, "config_file.json"))
    - config_file = os.path.abspath(os.path.join(current_dir, "config_file2.json"))

- If you don’t specify the db_name it generates all specifications for all schema.

```python
# Read from tables.json to get information of each schema
  complete_specs(
    dataset_path, # path to "../spider/tables.json"
    config_file,
    db_name="farm",
    num_query=1000
  )
```

```python
for _ in range(num): # num = num_query
    #...
    # randomlly choose one type for each attributes
    detail = {
      "meaningful_joins": type_of_join,
      "table_exp_type": table_exp_type,
      "where_type": where_type,
      "number_of_value_exp_in_group_by": group_by_type,
      "having_type": having_type,
      "orderby_type": orderby_type,
      "limit_type": limit_type,
      "value_exp_types": value_exp_type,
      "distinct_type": distinct_type,
      "min_max_depth_in_subquery": min_max_depth_in_subquery,
    }

    hash_value = calculate_hash(detail)
```

Query Generation

- If you want to test the query_generator with just one specification, specify the 'specs' parameter and pass it to the function. Alternatively, using 'None' for 'specs' prompts the query_generator to read all specifications from the file and create queries based on them. For testing you can copy specification from the specification files(such as farm.json in ../outpu/specs/farm.json)

Python

```python
if __name__ == "__main__":
    specs = {
        "farm": {
            "3aa777bab275f84f0459afb8b4192acdd5201c34": {
                "set_op_type": "none",
                "first_query": {
                    "meaningful_joins": "yes",
                    "table_exp_type": "subquery",
                    "where_type": {
                        "logical_operator": [
                            "AND",
                            "pattern_matching",
                            "not_exists_subquery",
                        ]
                    },
                    "number_of_value_exp_in_group_by": 2,
                    "having_type": {"single": "SUM"},
                    "orderby_type": "multiple",
                    "limit_type": "without_offset",
                    "value_exp_types": [
                        "arithmatic_exp_alias",
                        "string_func_exp_alias",
                    ],
                    "distinct_type": "none",
                    "min_max_depth_in_subquery": [1, 1],
                },
            }
        }
    }
    query_generator(
       db_name="farm", #If you don't set it, it generates for all schema
        specs=specs,
        max_num=1000,
        write_to_csv=True, # If you want to write results to CSV file
        random_choice=True, # If it is True it just return one query
    )
```