Flattening Semi-Structured Data

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

Structured data

```
  school_id  |   school_name    |  street_number  |  street_name  |  suffix  |      city      |  zip_code
------------ | ---------------- | --------------- | ------------- | -------- | -------------- | ----------
   s_19219   |  West Aurora HS  |       879       |     Main      |    St.   |  West Aurora   |   25041
   s_77465   |  Springtown HS   |      1645       |    Cherry     |    Rd.   |  Springtown    |   14556
```

```
  school_id  |                address_info
------------ | ------------------------------------------
   s_19219   |    {
             |        "school_name": "West Aurora HS",
             |        "address": {
             |            "street_number": 879,
             |            "street_name": "Main",
             |            "suffix":  "St."
             |            "city": "West Aurora",
             |            "zip_code": 25041
             |        }
             |    }
```

Semi-structured data

Data stored with braces in key-values pairs takes the data type `VARIANT`

```
{
    "school_name": "West Aurora HS",
    "address": {  -- Nested object
        "street_number": 879,
        "street_name": "Main",
        "suffix": "St.",
        "city": "West Aurora",
        "zip_code": 25041
    }
}
```

- Like a Python dictionary or JSON object
- Allows data to be stored in "raw" format
- Nest objects, like `address`
- Retrieve data in two different ways

Dot-notation

```
                 my_column
-------------------------------------------
       {
           "my_first_key": 2025,
           "my_second_key": {
               "a": "alpha",
               "b": "bravo"
           }
       }
```

```
SELECT
    my_column:my_first_key           -- Top-level
    my_column:my_second_key.a    -- Nested
    my_column:my_second_key.b    -- Nested
...
```

Makes it easy to retrieve top-level and **nested** values from `VARIANT` data

- Colon separates `<column-name>:<top-level-key>`
- Add a `.` followed by the nested field, `<column-name>:<top-level-key>.<nested-key>`
- Retrieve deeply-nested values

Dot-notation

```
SELECT
    address_info:school_name,                              -- Top-level, dot-notation

    address_info:address.street_number AS street_number,   -- Nested, dot-notation
    address_info:address.street_name AS street_name,
    address_info:address.suffix AS suffix

FROM SCHOOLS.school_info;
```

```
   school_name    |  street_number  |  street_name  |  suffix
----------------- | --------------- | ------------- | ---------
 West Aurora HS   |       879       |    Main       |    St.
 Springtown HS    |      1645       |    Cherry     |    Rd.
```

Bracket-notation

Provides an additional technique for retrieving top-level and nested values

- `<column-name>['<top-level-key']['...']`
- Many nested layers
- Like retrieving data from a Python dictionary
- Make sure to use **single quotes** (`'`)!

```
                        my_column
------------------------------------------------
         {
             "my_first_key": 2025,
              my_second_key": {
                 "a": "alpha",
                 "b": "bravo"
             }
         }
```

```sql
SELECT
    my_column['my_first_key'],          -- Top-level
    my_column['my_second_key']['a']      -- Nested
    my_column['my_second_key']['b']      -- Nested
...
```

Bracket-notation

```
SELECT
    address_info['school_name'],                          -- Top-level, bracket-notation

    address_info['address']['city'] AS city,               -- Nested, bracket-notation
    address_info['address']['zip_code'] AS zip_code

FROM SCHOOLS.school_info;
```

```
       school_name     |       city       |  zip_code
   ----------------- | -------------- | ----------
    West Aurora HS    |  West Aurora    |   25041
    Springtown HS     |  Springtown     |   14556
```

Transforming semi-structured data

```
SELECT
    school_id,

    address_info:school_name AS school_name,               -- Top-level, dot-notation

    address_info:address.street_number AS street_number,   -- Nested, dot-notation
    address_info:address.street_name AS street_name,
    address_info:address.suffix AS suffix,

    address_info['address']['city'] AS city,               -- Nested, bracket-notation
    address_info['address']['zip_code'] AS zip_code

FROM SCHOOLS.school_info;
```

Transforming semi-structured data

```
 school_id  |                address_info
----------- | ------------------------------------------
  s_19219   |    {
            |        "school_name": "West Aurora HS",
            |        "address": {
            |            "street_number": 879,
            |            "street_name": "Main",
            |            "suffix":  "St."
            |            "city": "West Aurora",
            |            "zip_code": 25041
            |        }
            |    }
```

```
 school_id  |    school_name    |  street_number  |  street_name  |  suffix   |      city      |  zip_code
----------- | ----------------- | --------------- | ------------- | --------- | -------------- | ----------
  s_19219   |  West Aurora HS   |       879       |    Main       |    St.    |  West Aurora   |   25041
  s_77465   |  Springtown HS    |      1645       |    Cherry     |    Rd.    |  Springtown    |   14556
```

Let's practice!

Multiple common table expressions

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

Jacob Roach
Field Data Engineer

Common table expressions (CTEs)

```sql
WITH seniors AS (
    SELECT
        student_id,
        first_name
    FROM STUDENTS.personal_info
    WHERE graduation_year = 2025
)

SELECT
    ...
FROM seniors
```

CTEs temporarily store the results of a query to eventually be used within another query

- Used to organize queries
- More readable and modular
- Not limited to a single CTE

Defining multiple CTEs

Multiple temporary results can be defined using a single `WITH` statement

- `...), <another-cte-name> AS (...)`
- Make table filtering and manipulation easier to understand
- `JOIN` multiple temporary result sets together
- Subqueries within another CTE

```
WITH <cte-name> AS (

    <query>

), <another-cte-name> AS (

    -- Add another query!
    <another-query>
)

-- These CTE's could be JOIN'd
SELECT ... ;
```

Joining temporary result sets

```sql
-- First common table expression
WITH seniors AS (
    SELECT student_id, first_name FROM STUDENTS.personal_info WHERE graduation_year = 2025

-- Second common table expression
), final_exam_grades AS (
    SELECT student_id, course_name, exam_score FROM STUDENTS.grades WHERE exam_type = 'Final'
)

SELECT
    seniors.first_name, final_exam_grades.course_name, final_exam_grades.exam_score
FROM final_exam_grades

-- Join the temporary result sets together
JOIN seniors ON final_exam_grades.student_id = seniors.student_id
```

Joining temporary result sets

```
 first_name  |  course_name  |  exam_score
------------ | ------------- | ------------
  Ryan       |  Calculus I   |      97
  Tatiana    |  Biology      |      98
  Pankaj     |  English III  |      92
  Taylor     |  Python       |      71
  Iris       |  Finance      |      89
  Charles    |  Marketing    |      88


                    ...
```

Using a temporary result in a CTE

```sql
WITH ny_schools AS (
    SELECT school_id, school_name, district FROM SCHOOL.school_info WHERE school_state = 'NY'
), ny_teachers AS (
    SELECT
        teacher_id, teacher_name, tenure, specialty
    FROM SCHOOLS.teachers
    WHERE school_id IN (SELECT school_id FROM ny_schools)  -- Filter by records in ny_schools
)

SELECT
    ny_teachers.teacher_name, ny_teachers.course_name, previous_courses.term
FROM SCHOOLS.previous_courses
JOIN ny_teachers ON previous_courses.teacher_id = ny_teachers.teacher_id AND
    previous_courses.course_area = ny_teachers.specialty
;
```

Using a temporary result in a CTE

```
WITH ny_schools AS (
    SELECT
        school_id,
        ...
    WHERE school_state = 'NY'
), ny_teachers AS (
    SELECT
        ...
    FROM SCHOOLS.teachers

    -- Filter by school_id in records above
    WHERE school_id IN (
        SELECT school_id FROM ny_schools
    )
)
```

- First, only filter for `school_id` 's in `NY`
- Temporary result stored in `ny_schools`
- In `ny_teachers` , filter records by `school_id` in `ny_schools`
- Could also `JOIN` , etc.

Matching teachers workloads to their specialties

```
...  -- Building the ny_teachers CTE

SELECT
    ny_teachers.teacher_name,
    ny_teachers.course_name,
    previous_courses.term
FROM SCHOOLS.previous_courses

-- Finally, join the CTE with the previous_courses table
JOIN ny_teachers ON
    previous_courses.teacher_id = ny_teachers.teacher_id AND
    previous_courses.course_area = ny_teachers.specialty
;
```

Matching teachers workloads to their specialties

```
    teacher_name      |       course_name         |  term
--------------------- | ------------------------- | -------
 Marcus Lee           |  U.S. History             |  H1'22
 Priya Desai          |  Algebra II               |  H2'22
 Elena Rodríguez      |  Environmental Science    |  H1'25
 Jamal Thompson       |  English I                |  H2'21
 Mei-Ling Chen        |  World History            |  H1'23
 Gregory O'Malley     |  Calculus II              |  H2'24


                                 ...
```

Let's practice!

Pivoting Data

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

Aggregated table

```
 course_name  |  exam_type  |  avg_exam_score
------------- | ----------- | ----------------
  Calculus I  |  Exam 1     |       81.78
  Calculus I  |  Exam 2     |       83.55
  Calculus I  |  Final      |       80.93

  Finance     |  Exam 1     |       89.47
  Finance     |  Exam 2     |       90.39
  Finance     |  Final      |       89.69

  Marketing   |  Exam 1     |       94.11
  Marketing   |  Exam 2     |       93.29
  Marketing   |  Final      |       93.81
```

"Pivoted" table

```
 course_name  |    "Exam 1"    |    "Exam 2"    |  "Final"
------------- | -------------- | -------------- | ---------
  Calculus I  |     81.78      |     83.55      |   80.93
  Finance     |     89.47      |     90.39      |   89.69
  Marketing   |     94.11      |     93.29      |   93.81
```

Creating a pivoted table

```
SELECT
    *
FROM SCHEMA.TABLE

PIVOT(
    -- Aggregation function
    SUM(<1>)

    -- Specify rows to pivot to columns
    FOR <2> IN (ANY ORDER BY <2>)

    -- No need to GROUP BY!
);
```

`PIVOT` offers a different way to output aggregated data by "pivoting" values into columns

- `SELECT *`, can use `EXCLUDE`
- `PIVOT` goes after `FROM ...`
- `ANY` values in `<2>`

`<1>` : field to aggregate

`<2>` : row values to turn into columns

CTE's and pivoted data

```
WITH exam_grades AS (
    SELECT
        ..., exam_score, exam_type
    FROM SCHEMA.TABLE
    WHERE ...
)

SELECT
    *   -- Could also use EXCLUDE
FROM exam_grades
PIVOT(
    AVG(exam_score)
    FOR exam_type IN (ANY ORDER BY exam_type)
);
```

First, define a CTE before using `PIVOT` !

`SELECT * FROM <cte>`

`exam_score` : field to aggregate

`exam_type` : row values to turn into columns

Comparing exam grades by type

```
WITH exam_grades AS (
    SELECT
        course_name, course_abbreviation, exam_score, exam_type
    FROM STUDENTS.grades
    WHERE course_level = '101'
)

SELECT
    * EXCLUDE course_abbreviation  -- Remove course_abbrevation from the result set
FROM exam_grades
PIVOT(
    AVG(exam_score)
    FOR exam_type IN (ANY ORDER BY exam_type)
);
```

Comparing exam grades by type

```
 course_name  |   exam_type   |  avg_exam_score
------------- | ------------- | ----------------
  Calculus I  |   Exam 1      |       81.78
  Calculus I  |   Exam 2      |       83.55
  Calculus I  |   Final       |       80.93

  Finance     |   Midterm 1   |       89.47
                     ...
```

```
 course_name  |    "Exam 1 "  |    "Exam 2"   |  "Final"
------------- | ------------- | ------------- | ---------
  Calculus I  |     81.78     |     83.55     |   80.93
  Finance     |     89.47     |     90.39     |   89.69
  Marketing   |     94.11     |     93.29     |   93.81
```

Let's practice!

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

Congratulations!

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

Jake Roach
Field Data Engineer

Thank you!

DATA TYPES AND FUNCTIONS IN SNOWFLAKE