

Context

SAMPLE command	2 – 3
TABLESAMPLE Command	4
Table Functions	4 – 8
Estimation functions	9 – 10
System Functions	11 – 12
User Defined Functions	12 – 14
Stored Procedure	14 – 17
Stream	18
Tasks	18 – 19
VARIANT Command	19 – 20
Array	21 – 22
Object	23 – 24
Flatten Array	25 – 26
Flatten Object	27 – 29
Extracting Values	29 – 32
Type Predicates	33 – 34
File Functions	34 – 36

SAMPLE command

1. Create a Table with the below query. And Insert the Data.

```
Query: CREATE or Replace TABLE employees (
    employee_id INT,
    first_name STRING,
    last_name STRING,
    department STRING,
    salary DECIMAL(10, 2)
);

INSERT INTO employees VALUES (1, 'John', 'Doe', 'Engineering', 75000);
INSERT INTO employees VALUES (2, 'Jane', 'Smith', 'Marketing', 65000);
INSERT INTO employees VALUES (3, 'Robert', 'Brown', 'Engineering', 80000);
INSERT INTO employees VALUES (4, 'Emily', 'Davis', 'Sales', 72000);
INSERT INTO employees VALUES (5, 'Michael', 'Wilson', 'Marketing', 70000);
INSERT INTO employees VALUES (6, 'Mary', 'Taylor', 'Engineering', 68000);
INSERT INTO employees VALUES (7, 'William', 'Anderson', 'Sales', 75000);
INSERT INTO employees VALUES (8, 'Patricia', 'Thomas', 'Marketing', 68000);
INSERT INTO employees VALUES (9, 'Charles', 'Jackson', 'Sales', 73000);
INSERT INTO employees VALUES (10, 'Linda', 'White', 'Engineering', 80000);
INSERT INTO employees VALUES (11, 'Daniel', 'Harris', 'Marketing', 69000);
INSERT INTO employees VALUES (12, 'Barbara', 'Martin', 'Sales', 71000);
INSERT INTO employees VALUES (13, 'Joseph', 'Thompson', 'Engineering', 77000);
INSERT INTO employees VALUES (14, 'Susan', 'Garcia', 'Marketing', 66000);
INSERT INTO employees VALUES (15, 'Thomas', 'Martinez', 'Sales', 74000);
INSERT INTO employees VALUES (16, 'Jessica', 'Robinson', 'Engineering', 79000);
INSERT INTO employees VALUES (17, 'Christopher', 'Clark', 'Marketing', 67000);
INSERT INTO employees VALUES (18, 'Sarah', 'Rodriguez', 'Sales', 71000);
INSERT INTO employees VALUES (19, 'David', 'Lewis', 'Engineering', 75000);
INSERT INTO employees VALUES (20, 'Karen', 'Lee', 'Marketing', 64000);
INSERT INTO employees VALUES (21, 'James', 'Walker', 'Sales', 73000);
INSERT INTO employees VALUES (22, 'Nancy', 'Hall', 'Engineering', 80000);
INSERT INTO employees VALUES (23, 'George', 'Allen', 'Marketing', 65000);
INSERT INTO employees VALUES (24, 'Donna', 'Young', 'Sales', 72000);
INSERT INTO employees VALUES (25, 'Paul', 'Hernandez', 'Engineering', 68000);
INSERT INTO employees VALUES (26, 'Laura', 'King', 'Marketing', 67000);
INSERT INTO employees VALUES (27, 'Mark', 'Wright', 'Sales', 74000);
INSERT INTO employees VALUES (28, 'Michelle', 'Lopez', 'Engineering', 79000);
INSERT INTO employees VALUES (29, 'Steven', 'Hill', 'Marketing', 66000);
INSERT INTO employees VALUES (30, 'Dorothy', 'Scott', 'Sales', 71000);
INSERT INTO employees VALUES (31, 'Edward', 'Green', 'Engineering', 75000);
INSERT INTO employees VALUES (32, 'Sarah', 'Adams', 'Marketing', 64000);
INSERT INTO employees VALUES (33, 'Kevin', 'Baker', 'Sales', 72000);
INSERT INTO employees VALUES (34, 'Angela', 'Gonzalez', 'Engineering', 68000);
INSERT INTO employees VALUES (35, 'Jason', 'Nelson', 'Marketing', 67000);
INSERT INTO employees VALUES (36, 'Helen', 'Carter', 'Sales', 74000);
INSERT INTO employees VALUES (37, 'Brian', 'Mitchell', 'Engineering', 79000);
INSERT INTO employees VALUES (38, 'Sandra', 'Perez', 'Marketing', 66000);
```

```

INSERT INTO employees VALUES (39, 'Scott', 'Roberts', 'Sales', 71000);
INSERT INTO employees VALUES (40, 'Megan', 'Turner', 'Engineering', 75000);
INSERT INTO employees VALUES (41, 'Richard', 'Phillips', 'Marketing', 64000);
INSERT INTO employees VALUES (42, 'Patricia', 'Campbell', 'Sales', 72000);
INSERT INTO employees VALUES (43, 'Frank', 'Parker', 'Engineering', 68000);
INSERT INTO employees VALUES (44, 'Betty', 'Evans', 'Marketing', 67000);
INSERT INTO employees VALUES (45, 'Joseph', 'Edwards', 'Sales', 74000);
INSERT INTO employees VALUES (46, 'Emily', 'Collins', 'Engineering', 79000);
INSERT INTO employees VALUES (47, 'Paul', 'Stewart', 'Marketing', 66000);
INSERT INTO employees VALUES (48, 'Amanda', 'Sanchez', 'Sales', 71000);
INSERT INTO employees VALUES (49, 'Kenneth', 'Morris', 'Engineering', 75000);
INSERT INTO employees VALUES (50, 'Linda', 'Rogers', 'Marketing', 64000);

```

"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾

Code Versions

```

1 CREATE or Replace TABLE employees (
2     employee_id INT,
3     first_name STRING,
4     last_name STRING,
5     department STRING,
6     salary DECIMAL(10, 2)
7 );
8
9 INSERT INTO employees VALUES (1, 'John', 'Doe', 'Engineering', 75000);
10 INSERT INTO employees VALUES (2, 'Jane', 'Smith', 'Marketing', 65000);
11 INSERT INTO employees VALUES (3, 'Robert', 'Brown', 'Engineering', 80000);
12 INSERT INTO employees VALUES (4, 'Emily', 'Davis', 'Sales', 72000);
13 INSERT INTO employees VALUES (5, 'Michael', 'Wilson', 'Marketing', 70000);
14 INSERT INTO employees VALUES (6, 'Mary', 'Taylor', 'Engineering', 68000);
15 INSERT INTO employees VALUES (7, 'William', 'Anderson', 'Sales', 75000);
16 INSERT INTO employees VALUES (8, 'Patricia', 'Thomas', 'Marketing', 68000);
17 INSERT INTO employees VALUES (9, 'Charles', 'Jackson', 'Sales', 73000);
18 INSERT INTO employees VALUES (10, 'Linda', 'White', 'Engineering', 80000);
19 INSERT INTO employees VALUES (11, 'Daniel', 'Harris', 'Marketing', 69000);

```

↳ Results ↵ Chart

	number of rows inserted
1	1

Query Details ...
Query duration 346ms
Rows 1
Query ID 01b51b5a-0001-2587-0...

2. Fraction-based sampling (10%)

Query: `SELECT * FROM employees SAMPLE (10);`

3. Approximately 10% of the rows (5 rows) selected randomly.

"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾

Code Versions

```

1 | SELECT * FROM employees SAMPLE (10);

```

↳ Results ↵ Chart

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY
1	8	Patricia	Thomas	68000.00
2	32	Sarah	Adams	64000.00
3	40	Megan	Turner	75000.00
4	43	Frank	Parker	68000.00
5	47	Paul	Stewart	66000.00

Query Details ...
Query duration 128ms
Rows 5
Query ID 01b51b5c-0001-2587-0...

TABLESAMPLE Command

1. Fixed-size sampling (5 rows).

Query: SELECT * FROM employees TABLESAMPLE BERNOUILLI (5 ROWS);

2. Exactly 5 rows selected randomly using Bernoulli sampling.

The screenshot shows a Snowflake query results page. The query executed was:

```
1 | SELECT * FROM employees TABLESAMPLE BERNOUILLI (5 ROWS);
```

The results table has columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, DEPARTMENT, and SALARY. The data is as follows:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY
1	27	Mark	Wright	Sales	74000.00
2	41	Richard	Phillips	Marketing	64000.00
3	3	Robert	Brown	Engineering	80000.00
4	19	David	Lewis	Engineering	75000.00
5	30	Dorothy	Scott	Sales	71000.00

On the right side of the results table, there is a "Query Details" panel. It shows the following information:

- Query duration: 154ms
- Rows: 5
- Query ID: 01b51b5d-0001-2586-...

Table Functions

1. The SPLIT_TO_TABLE function splits a string into a table of rows. We'll use this function to split a comma-separated string.
2. Creating a comma-separated string of employee names.

Query: SELECT LISTAGG(first_name || ' ' || last_name, ',') WITHIN GROUP (ORDER BY employee_id) AS names
FROM employees

The screenshot shows a Snowflake query results page. The query executed was:

```
1 | SELECT LISTAGG(first_name || ' ' || last_name, ',') WITHIN GROUP (ORDER BY employee_id) AS names
2 | FROM employees
```

The results table has a single column named "NAMES". The data is as follows:

NAMES
John Doe,Jane Smith,Robert Brown,Emily Davis,Michael Wilson,Mary Taylor,William Anderson,Patricia Thomas,Charles Jackson,Lin

On the right side of the results table, there is a "Query Details" panel. It shows the following information:

- Query duration: 32ms

3. Using SPLIT_TO_TABLE to split the string into separate rows.

Query: SELECT value AS employee_name
FROM EMPLOYEES,
TABLE(SPLIT_TO_TABLE(FIRST_NAME, ','));

The screenshot shows the Snowflake UI interface. In the top-left, the connection is set to "SAMPLE_SNOW"."Snowflake". On the right, there are "Code Versions" and a search icon. Below the connection, the query is displayed:

```
1 | SELECT value AS employee_name
2 | FROM EMPLOYEES,
3 | TABLE(SPLIT_TO_TABLE(FIRST_NAME, ', '));
```

Below the query, the results are shown in a table:

EMPLOYEE_NAME
1 John
2 Jane
3 Robert
4 Emily
5 Michael
6 Mary
7 William
8 Patricia
9 Charles
10 Linda
11 Daniel
12 Barbara

On the right side of the results table, there is a "Query Details" panel:

- Query duration: 172ms
- Rows: 50
- Query ID: 01b51b61-0001-2586-0...

Below the results table, there is a smaller table showing the distribution of names:

EMPLOYEE_NAME	A
Emily	2
Patricia	2
Linda	2
+ 41 more	

4. Using GENERATOR to create a series of numbers.

Query: SELECT
SEQ4() AS n
FROM
TABLE(GENERATOR(ROWCOUNT => 10))

The screenshot shows the Snowflake UI interface. In the top-left, the connection is set to "SAMPLE_SNOW"."Snowflake". On the right, there are "Code Versions" and a search icon. Below the connection, the query is displayed:

```
1 | SELECT
2 | SEQ4() AS n
3 | FROM
4 | TABLE(GENERATOR(ROWCOUNT => 10))
```

Below the query, the results are shown in a table:

N
1
2
3
4
5
6
7
8
9
10

On the right side of the results table, there is a "Query Details" panel:

- Query duration: 590ms
- Rows: 10
- Query ID: 01b51b6f-0001-2587-0...

Below the results table, there is a histogram showing the distribution of values:

N	#
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1

3. Using the generated series to create employee data.

Query: SELECT
 EMPLOYEE_ID,
 FIRST_NAME || LAST_NAME AS Full_Name,
 CASE
 WHEN EMPLOYEE_ID % 2 = 0 THEN 'Engineering'
 WHEN EMPLOYEE_ID % 3 = 0 THEN 'Marketing'
 ELSE 'Sales'
 END AS department,
 50000 + (EMPLOYEE_ID * 1000) AS salary
 FROM
 EMPLOYEES;

```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
Code Versions Q
SELECT
    EMPLOYEE_ID,
    FIRST_NAME || LAST_NAME AS Full_Name,
    CASE
        WHEN EMPLOYEE_ID % 2 = 0 THEN 'Engineering'
        WHEN EMPLOYEE_ID % 3 = 0 THEN 'Marketing'
        ELSE 'Sales'
    END AS department,
    50000 + (EMPLOYEE_ID * 1000) AS salary
FROM
    EMPLOYEES;
1 2 3 4 5 6 7 8 9 10 11
```

	EMPLOYEE_ID	FULL_NAME	DEPARTMENT	SALARY
1	1	JohnDoe	Sales	51000
2	2	JaneSmith	Engineering	52000
3	3	RobertBrown	Marketing	53000
4	4	EmilyDavis	Engineering	54000
5	5	MichaelWilson	Sales	55000
6	6	MaryTaylor	Engineering	56000
7	7	WilliamAnderson	Sales	57000

Query Details ...

Query duration 318ms

Rows 50

Query ID 01b51b72-0001-2586-0...

EMPLOYEE_ID #

4. Creating a User-Defined Function (UDF) and using it in conjunction with a table function.
 5. Create a UDF to calculate a bonus for each employee.

Query: CREATE OR REPLACE FUNCTION calculate_bonus(salary DECIMAL)
 RETURNS DECIMAL
 LANGUAGE SQL
 AS 'salary * 0.10';

```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
Code Versions Q
CREATE OR REPLACE FUNCTION calculate_bonus(salary DECIMAL)
RETURNS DECIMAL
LANGUAGE SQL
AS 'salary * 0.10';
1 2 3 4
```

	status
1	Function CALCULATE_BONUS successfully created.

Query Details ...

Query duration 52ms

6. Use this UDF in a query to display the bonus along with employee details.

Query: SELECT

```

    employee_id,
    first_name,
    last_name,
    department,
    salary,
    calculate_bonus(salary) AS bonus
FROM
    employees;
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	BONUS
1	1	John	Doe	Engineering	75000.00	7500.00
2	2	Jane	Smith	Marketing	65000.00	6500.00
3	3	Robert	Brown	Engineering	80000.00	8000.00
4	4	Emily	Davis	Sales	72000.00	7200.00
5	5	Michael	Wilson	Marketing	70000.00	7000.00
6	6	Mary	Taylor	Engineering	68000.00	6800.00
7	7	William	Anderson	Sales	75000.00	7500.00

7. The ARRAY_AGG function aggregates rows into an array.
 8. Aggregate employee names into an array based on their department.
 9. Use ARRAY_AGG with GROUP BY to create an array of employee names for each department.

Query: SELECT

```

    department,
    ARRAY_AGG(first_name || ' ' || last_name) AS employees
FROM
    employees
GROUP BY
    department;
```

	DEPARTMENT	EMPLOYEES
1	Sales	["Emily Davis", "William Anderson", "Charles Jackson", "Barbara Martin", "Thomas Martinez", "Jane Smith", "Michael Wilson", "Patricia Thomas", "Daniel Harris", "Susan Garcia", "Christopher Lee"]
2	Engineering	["John Doe", "Robert Brown", "Mary Taylor", "Linda White", "Joseph Thompson", "Jessica Robin"]
3	Marketing	["Jane Smith", "Michael Wilson", "Patricia Thomas", "Daniel Harris", "Susan Garcia", "Christopher Lee"]

10. The DATEADD function adds or subtracts an interval to a date.
11. Use LATERAL VIEW to join the employees table with the DATEADD function.
12. Add a specific interval (e.g., days) to a date.

Query: SELECT

```
e.employee_id,
e.first_name,
e.last_name,
e.department,
e.salary,
DATEADD(DAY, e.employee_id, CURRENT_DATE()) AS future_date
FROM
employees e
ORDER BY
e.employee_id;
```

The screenshot shows a Snowflake query interface. The code area contains the provided SQL query. The results section displays a table with columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, DEPARTMENT, SALARY, and FUTURE_DATE. The data is as follows:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	FUTURE_DATE
1	1	John	Doe	Engineering	75000.00	2024-06-19
2	2	Jane	Smith	Marketing	65000.00	2024-06-20
3	3	Robert	Brown	Engineering	80000.00	2024-06-21
4	4	Emily	Davis	Sales	72000.00	2024-06-22
5	5	Michael	Wilson	Marketing	70000.00	2024-06-23
6	6	Mary	Taylor	Engineering	68000.00	2024-06-24
7	7	William	Anderson	Sales	75000.00	2024-06-25

On the right side, there are two cards: 'Query Details' showing duration (1.7s), rows (50), and query ID (01b51b9b-0001-2586...); and a histogram for 'EMPLOYEE_ID' ranging from 1 to 50.

Estimation functions

1. The APPROX_COUNT_DISTINCT function provides an approximate count of distinct values in a column.
2. It is faster than COUNT(DISTINCT column_name) and is useful for large datasets.

Query: `SELECT APPROX_COUNT_DISTINCT(department) AS approx_distinct_departments FROM employees;`

3. This query returns an approximate count of distinct departments in the employee's table.
4. Since we inserted employees in three departments (Engineering, Marketing, and Sales), the approximate distinct count is expected to be close to 3.

The screenshot shows a Snowflake query interface. The query is:

```
1 | SELECT APPROX_COUNT_DISTINCT(department) AS approx_distinct_departments FROM employees;
```

The results table has one row with the following data:

APPROX_DISTINCT_DEPARTMENTS
3

On the right side, there is a "Query Details" panel showing:

- Query duration: 1.4s
- Rows: 1

5. The APPROX_PERCENTILE function provides an approximate percentile value of a numeric column.

Query: `SELECT APPROX_PERCENTILE(salary, 0.50) AS approx_median_salary FROM employees;`

6. This query returns the approximate median salary of employees. Given our inserted salaries, the median salary should be around 71000.

The screenshot shows a Snowflake query interface. The query is:

```
1 | SELECT APPROX_PERCENTILE(salary, 0.50) AS approx_median_salary FROM employees;
```

The results table has one row with the following data:

APPROX_MEDIAN_SALARY
71000

The screenshot shows a Snowflake query interface. The query is:

```
1 | SELECT APPROX_PERCENTILE(salary, 0.50) AS approx_median_salary FROM employees;
```

The results table has one row with the following data:

APPROX_MEDIAN_SALARY
71000

On the right side, there is a "Query Details" panel showing:

- Query duration

7. HyperLogLog is an algorithm used to estimate the number of distinct elements in a set, which can be used via HLL and HLL_ACCUMULATE.

Query: `SELECT HLL_ESTIMATE(HLL_ACCUMULATE(department)) AS approx_distinct_departments FROM employees;`

8. This query uses HyperLogLog to estimate the number of distinct departments in the employees table, which should be close to 3.



The screenshot shows a Snowflake query results page. The query is:

```
1 | SELECT HLL_ESTIMATE(HLL_ACCUMULATE(department)) AS approx_distinct_departments FROM employees;
```

The results table has one row with the column name `APPROX_DISTINCT_DEPARTMENTS` and value 3. The sidebar on the right shows "Query duration 150ms".

9. Initialize HLL and estimate unique employee IDs.

Query: `SELECT HLL_ESTIMATE(HLL_ACCUMULATE(employee_id)) AS approx_distinct_employee_ids FROM employees;`

10. This query uses HyperLogLog to estimate the number of distinct employee IDs in the employees table.
11. Since there are 50 unique employees, the approximate distinct count is expected to be close to 50.



The screenshot shows a Snowflake query results page. The query is:

```
1 | SELECT HLL_ESTIMATE(HLL_ACCUMULATE(employee_id)) AS approx_distinct_employee_ids FROM employees;
```

The results table has one row with the column name `APPROX_DISTINCT_EMPLOYEE_IDS` and value 50. The sidebar on the right shows "Query duration 163ms".

System Functions

1. Go to Query history and get the Query ID and Session ID.

The screenshot shows the Snowflake interface. On the left, a sidebar menu includes 'Create', 'Search', 'Projects', 'Data', 'Data Products', 'AI & ML', 'Monitoring', 'Query History' (which is selected), 'Copy History', 'Task History', 'Dynamic Tables', 'Trust Center', 'Governance', and 'Admin'. The main area is titled 'Query - 01b51b9b-0001-2586-0000-0004e88e4bfd'. It displays 'COMPUTE_WH' and 'SNOWFLAKE97098' under 'Query Details'. The 'Query Details' tab is active, showing the following information:

Details	
Status	Success
Duration	1.7s
Driver Status	Supported
Start Time	6/19/2024, 11:49:33 AM
Query ID	01b51b9b-0001-2586-0000-0004e88e4... (link)
Client Driver	Go 1.1.5 (link)
End Time	6/19/2024, 11:49:35 AM
Query Tag	—
Session ID	21081501829 (link)
Warehouse Size	X-Small

2. Cancel the query that you specified by id.

Query: `SELECT SYSTEM$CANCEL_QUERY('01b51b9b-0001-2586-0000-0004e88e4bfd')`

3. Right now we are not executing this query. It will show Identified SQL statement is not currently executing.

The screenshot shows the Snowflake results page. The query entered is `SELECT SYSTEM$CANCEL_QUERY('01b51b9b-0001-2586-0000-0004e88e4bfd');`. The results table shows one row with the message 'Identified SQL statement is not currently executing.' To the right, the 'Query Details' section shows a duration of 20ms.

4. Abort session that you specified by id.

Query: `SELECT SYSTEM$ABORT_SESSION('21081501829');`

The screenshot shows the Snowflake results page. The query entered is `SELECT SYSTEM$ABORT_SESSION('21081501829');`. The results table shows one row with the value 'null'. To the right, the 'Query Details' section shows a duration of 22ms.

- Get the type of the object.

Query: `SELECT SYSTEM$TYPEOF('EMPLOYEES');`

The screenshot shows a Snowflake query editor with the following details:

- Connection: "SAMPLE_SNOW"."Snowflake"
- Settings: Various options like "Code Versions", "Query Details", and "Query duration".
- Query: `SELECT SYSTEM$TYPEOF('EMPLOYEES');`
- Results:

SYSTEM\$TYPEOF('EMPLOYEES')
VARCHAR(9)[LOB]

User Defined Functions

- This UDF categorizes employees into "High Salary" and "Low Salary" based on a predefined threshold (let's set it at \$70,000).

Query: `CREATE OR REPLACE FUNCTION employee_status(salary DECIMAL)
RETURNS STRING
LANGUAGE SQL
AS '
CASE
WHEN salary >= 70000 THEN "High Salary"
ELSE "Low Salary"
END
';`

The screenshot shows a Snowflake query editor with the following details:

- Connection: "SAMPLE_SNOW"."Snowflake"
- Settings: Various options like "Code Versions", "Query Details", and "Query duration".
- Query: `CREATE OR REPLACE FUNCTION employee_status(salary DECIMAL)
RETURNS STRING
LANGUAGE SQL
AS '
CASE
WHEN salary >= 70000 THEN ''High Salary''
ELSE ''Low Salary''
END
';`
- Results:

status
Function EMPLOYEE_STATUS successfully created.

2. Determine employee status based on salary.

Query: `SELECT first_name, last_name, salary, employee_status(salary) AS status FROM employees;`

The screenshot shows a Snowflake query results page. The query selected is:

```
1 | SELECT first_name, last_name, salary, employee_status(salary) AS status
2 | FROM employees;
```

The results table has columns: FIRST_NAME, LAST_NAME, SALARY, and STATUS. The data is as follows:

	FIRST_NAME	LAST_NAME	SALARY	STATUS
1	John	Doe	75000.00	High Salary
2	Jane	Smith	65000.00	Low Salary
3	Robert	Brown	80000.00	High Salary
4	Emily	Davis	72000.00	High Salary
5	Michael	Wilson	70000.00	High Salary
6	Mary	Taylor	68000.00	Low Salary
7	William	Anderson	75000.00	High Salary
8	Patricia	Thomas	68000.00	Low Salary
9	Charles	Jackson	73000.00	High Salary
10	Linda	White	80000.00	High Salary
11	Daniel	Harris	69000.00	Low Salary
12	Barbara	Martin	71000.00	High Salary

On the right side, there is a sidebar titled "Query Details" showing the duration (1.5s), rows (50), and query ID (01b51bbe-0001-2586-0...). Below it are filters for FIRST_NAME and LAST_NAME.

3. This UDF concatenates an employee's first name and last name with a space in between.

Query: `CREATE OR REPLACE FUNCTION full_name(first_name STRING, last_name STRING)
RETURNS STRING
LANGUAGE SQL
AS '
 first_name || ' ' || last_name
';`

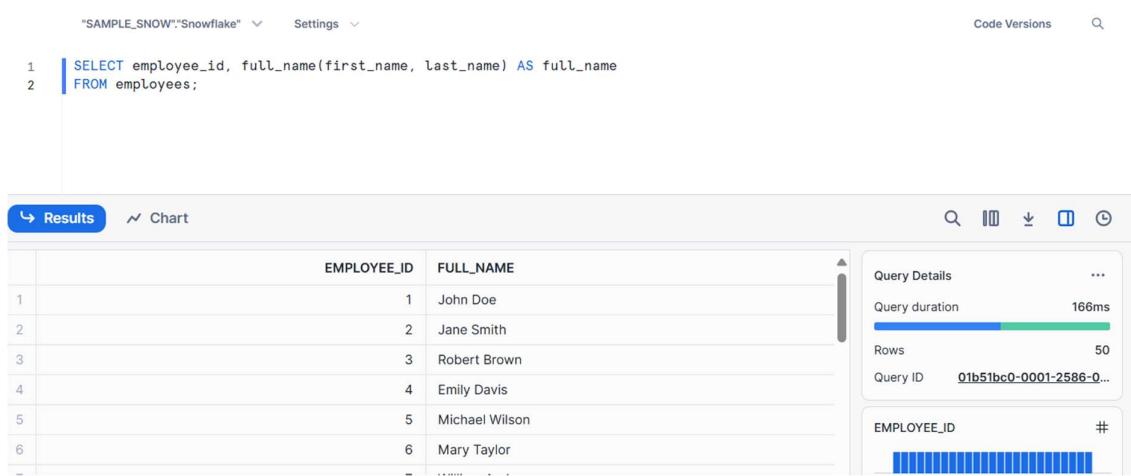
The screenshot shows a Snowflake query results page. The query selected is:

```
1 | CREATE OR REPLACE FUNCTION full_name(first_name STRING, last_name STRING)
2 | RETURNS STRING
3 | LANGUAGE SQL
4 | AS '
5 |     first_name || ' ' || last_name
6 | ';
```

The results table has a single column "status" with the message: "Function FULL_NAME successfully created." On the right side, there is a sidebar titled "Query Details" showing the duration (51ms) and rows (1).

- Concatenate first name and last name.

Query: `SELECT employee_id, full_name(first_name, last_name) AS full_name
FROM employees;`



The screenshot shows a Snowflake query results interface. At the top, there's a code editor with the following SQL query:

```

1 | SELECT employee_id, full_name(first_name, last_name) AS full_name
2 | FROM employees;

```

Below the code editor is a results table with two columns: `EMPLOYEE_ID` and `FULL_NAME`. The data is as follows:

	EMPLOYEE_ID	FULL_NAME
1	1	John Doe
2	2	Jane Smith
3	3	Robert Brown
4	4	Emily Davis
5	5	Michael Wilson
6	6	Mary Taylor

On the right side of the results table, there's a sidebar titled "Query Details" showing the following metrics:

- Query duration: 166ms
- Rows: 50
- Query ID: 01b51bc0-0001-2586-0...

Stored Procedure

- Create a Customer table.

Query: `CREATE OR REPLACE TABLE customer (customer_id INT,
salutation VARCHAR,
first_name VARCHAR,
last_name VARCHAR,
birth_day INT,
birth_month INT,
birth_year INT,
birth_country VARCHAR,
email_address VARCHAR,
cust_status VARCHAR);`

`INSERT INTO customer (customer_id, salutation, first_name, last_name, birth_day,
birth_month, birth_year, birth_country, email_address, cust_status)
VALUES
(101, 'Mr.', 'Cust-1', 'LName-1', 10, 12, 2000, 'Japan', 'cust-1.lname-1@gmail.com', 'Active'),
(102, 'Mr.', 'Cust-2', 'LName-2', 27, 11, 1999, 'USA', 'cust-2.lname-2@gmail.com', 'Inactive'),
(103, 'Mr.', 'Cust-3', 'LName-3', 21, 2, 1998, 'UK', 'cust-3.lname-3@gmail.com', 'Blocked'),
(104, 'Mr.', 'Cust-4', 'LName-4', 19, 9, 1997, 'USA', 'cust-4.lname-4@gmail.com', 'Active'),
(105, 'Mr.', 'Cust-5', 'LName-5', 11, 3, 1997, 'Canada', 'cust-5.lname-5@gmail.com',
'Unknown');`

2. Check the query.

```

CREATE OR REPLACE TABLE customer (
    customer_id INT,
    salutation VARCHAR,
    first_name VARCHAR,
    last_name VARCHAR,
    birth_day INT,
    birth_month INT,
    birth_year INT,
    birth_country VARCHAR,
    email_address VARCHAR,
    cust_status VARCHAR
);

INSERT INTO customer (customer_id, salutation, first_name, last_name, birth_day, birth_month, birth_year, birth_country, email_address, cust_status)
VALUES
(101, 'Mr.', 'LName-1', '10', 12, 2000, 'Japan', 'cust-1.lname-1@gmail.com', 'Active'),
(102, 'Mr.', 'LName-2', '11', 11, 1999, 'USA', 'cust-2.lname-2@gmail.com', 'Inactive')

```

Results

number of rows inserted	Query Details	...
5	Query duration	1.6s

3. Create a Stored procedure that deletes rows where cust_status is 'Inactive' from the customer table.

Query: CREATE OR REPLACE PROCEDURE purge_data()

```

RETURNS VARCHAR
LANGUAGE SQL
AS
$$
DECLARE
    message VARCHAR;
BEGIN
    DELETE FROM customer WHERE cust_status = 'Inactive';
    message := 'Inactive customer data deleted successfully';
    RETURN message;
END;
$$;

```

```

1 CREATE OR REPLACE PROCEDURE purge_data()
2     RETURNS VARCHAR
3     LANGUAGE SQL
4 AS
5 $$
6 DECLARE
7     message VARCHAR;
8 BEGIN
9     DELETE FROM customer WHERE cust_status = 'Inactive';
10    message := 'Inactive customer data deleted successfully';
11    RETURN message;
12 END;
13 $$;
14

```

Results

status	Query Details	...
Function PURGE_DATA successfully created.	Query duration	66ms

4. Calling purge_data Stored Procedure.
5. This statement calls the purge_data stored procedure, which in turn deletes rows where cust_status is 'Inactive' from the customer table.

Query: CALL purge_data();

```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
```

```
1 | CALL purge_data();
```

PURGE_DATA
1 Inactive customer data deleted successfully

Query Details
...

Query duration
574ms

6. Check the data.

```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
```

```
1 | Select * from CUSTOMER
```

ME	BIRTH_DAY	BIRTH_MONTH	BIRTH_YEAR	BIRTH_COUNTRY	EMAIL_ADDRESS	CUST_STATUS
1	10	12	2000	Japan	cust-1.lname-1@gmail.com	Active
2	21	2	1998	UK	cust-3.lname-3@gmail.com	Blocked
3	19	9	1997	USA	cust-4.lname-4@gmail.com	Active
4	11	3	1997	Canada	cust-5.lname-5@gmail.com	Unknown

Query Details
...

Query duration
1.2s

Rows
4

Query ID
01b51c1f-0001-2586-0...

7. Create a Stored Procedure that deletes rows from the customer table where cust_status is 'Unknown'.

Query: CREATE OR REPLACE PROCEDURE purge_data_by_status(in_status VARCHAR)

RETURNS VARCHAR

LANGUAGE SQL

AS

\$\$

DECLARE

message VARCHAR;

BEGIN

DELETE FROM customer WHERE cust_status = :in_status;

message := in_status || ' customer data deleted successfully';

RETURN message;

END;

\$\$;

8. Check the data.

The screenshot shows the Snowflake UI interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is a code editor window containing the following SQL code:

```

1 CREATE OR REPLACE PROCEDURE purge_data_by_status(in_status VARCHAR)
2   RETURNS VARCHAR
3   LANGUAGE SQL
4   AS
5   $$ 
6   DECLARE
7     message VARCHAR;
8   BEGIN
9     DELETE FROM customer WHERE cust_status = :in_status;
10    message := in_status || ' customer data deleted successfully';
11    RETURN message;
12  END;
13  $$;
14

```

Below the code editor is a results table with one row:

	status
1	Function PURGE_DATA_BY_STATUS successfully created.

On the right side of the results table, there are "Query Details" and "Query duration" sections. The "Query duration" bar is green and labeled "46ms".

9. Calling `purge_data_by_status` Stored Procedure with Parameter.
10. This statement calls the `purge_data_by_status` stored procedure with 'Unknown' as the input `in_status`.
11. The procedure then deletes rows from the customer table where `cust_status` is 'Unknown'.

Query: `CALL purge_data_by_status('Unknown');`

The screenshot shows the Snowflake UI interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is a code editor window containing the following SQL code:

```

1 CALL purge_data_by_status('Unknown');

```

Below the code editor is a results table with one row:

	PURGE_DATA_BY_STATUS
1	Unknown customer data deleted successfully

On the right side of the results table, there are "Query Details" and "Query duration" sections. The "Query duration" bar is green and labeled "754ms".

6. Check the data.

The screenshot shows the Snowflake UI interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is a code editor window containing the following SQL code:

```

1 Select *FROM CUSTOMER

```

Below the code editor is a results table with three rows of data:

	NAME	BIRTH_DAY	BIRTH_MONTH	BIRTH_YEAR	BIRTH_COUNTRY	EMAIL_ADDRESS	CUST_STATUS
1	-1	10	12	2000	Japan	cust-1.lname-1@gmail.com	Active
2	-3	21	2	1998	UK	cust-3.lname-3@gmail.com	Blocked
3	-4	19	9	1997	USA	cust-4.lname-4@gmail.com	Active

On the right side of the results table, there are "Query Details" and "Query duration" sections. The "Query duration" bar is blue and labeled "176ms". There is also a "Rows" section indicating 3 rows and a "Query ID" section with the value "01b51c22-0001-2587-0...".

Stream

1. Creating a stream.

Query: CREATE OR REPLACE STREAM employee_stream ON TABLE employees;

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW" and "Snowflake". Below it is a code editor window containing the SQL command: "CREATE OR REPLACE STREAM employee_stream ON TABLE employees;". To the right of the code editor is a "Results" tab, which is currently selected. The results table has one row with the status message: "Stream EMPLOYEE_STREAM successfully created.". On the far right, there's a "Query Details" panel showing a green progress bar for "Query duration" at 223ms.

2. Changes to the employees table since the stream was created.

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW" and "Snowflake". Below it is a code editor window containing the SQL command: "SELECT * FROM employee_stream;". To the right of the code editor is a "Results" tab, which is currently selected. The results table has a header row with columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, DEPARTMENT, SALARY, and METADATA\$ACTION. Below the header, a message says "Query produced no results". On the far right, there's a "Query Details" panel showing a green progress bar for "Query duration" at 153ms, and a note that "Rows" are 0. The "Query ID" is listed as 01b51bd8-0001-2587-0... .

Tasks

1. Task created and scheduled to run daily.
2. Insert a summary of the total salary per department into employee_summary.

Query: CREATE OR REPLACE TASK daily_employee_summary

WAREHOUSE = COMPUTE_WH
SCHEDULE = 'USING CRON 0 0 * * * UTC'
AS

INSERT INTO employee_summary (department, total_salary)
SELECT department, SUM(salary) FROM employees GROUP BY department;

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW" and "Snowflake". Below it is a code editor window containing the SQL command for creating a task:
1. CREATE OR REPLACE TASK daily_employee_summary
2. WAREHOUSE = COMPUTE_WH
3. SCHEDULE = 'USING CRON 0 0 * * * UTC'
4. AS
5. INSERT INTO employee_summary (department, total_salary)
6. SELECT department, SUM(salary) FROM employees GROUP BY department;

To the right of the code editor is a "Results" tab, which is currently selected. The results table has one row with the status message: "Task DAILY_EMPLOYEE_SUMMARY successfully created.". On the far right, there's a "Query Details" panel showing a green progress bar for "Query duration" at 75ms, and a note that "Rows" are 1.

3. Enabling the Task.

Query: ALTER TASK daily_employee_summary RESUME;

The screenshot shows the Snowflake UI interface. At the top, it displays the connection name "SAMPLE_SNOW"."Snowflake" and various navigation options like "Settings" and "Code Versions". Below the connection bar, a single-line query is shown: "ALTER TASK daily_employee_summary RESUME;". In the main area, there's a "Results" tab selected, showing a table with one row labeled "status". The status message reads "Statement executed successfully.". To the right of the results table, a "Query Details" panel is visible, showing a progress bar for "Query duration" (117ms) and a count of "Rows" (1). There are also icons for search, refresh, and other operations.

VARIANT Command

In Snowflake, the VARIANT data type is a flexible and powerful way to store semi-structured and nested data within a single column of a table. It is designed to handle data that doesn't fit neatly into traditional tabular structures, such as JSON, XML, or other complex hierarchical data.

1. Create a table and insert a value.

Query: CREATE TABLE varia (float1 FLOAT, v VARIANT, float2 FLOAT);

INSERT INTO varia (float1, v, float2) VALUES (1.23, NULL, NULL);

The screenshot shows the Snowflake UI interface. At the top, it displays the connection name "SAMPLE_SNOW"."Snowflake" and various navigation options like "Settings" and "Code Versions". Below the connection bar, two queries are shown: "CREATE TABLE varia (float1 FLOAT, v VARIANT, float2 FLOAT);" and "INSERT INTO varia (float1, v, float2) VALUES (1.23, NULL, NULL);". In the main area, there's a "Results" tab selected, showing a table with one row labeled "number of rows inserted". The value is 1. To the right of the results table, a "Query Details" panel is visible, showing a progress bar for "Query duration" (1.6s).

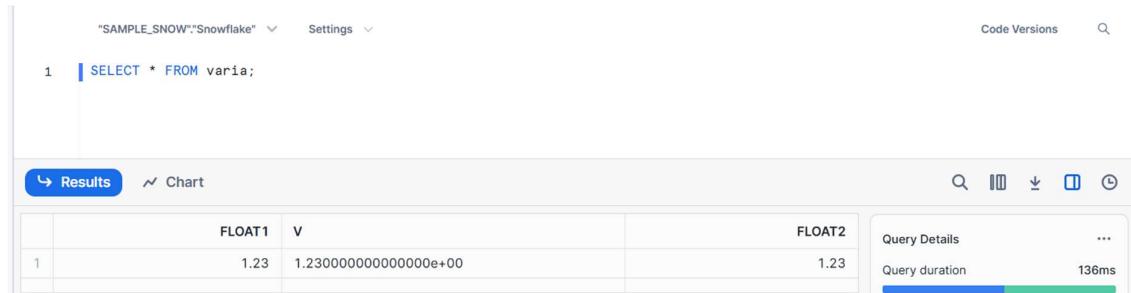
2. The first UPDATE converts a value from a FLOAT to a VARIANT. The second UPDATE converts a value from a VARIANT to a FLOAT.

Query: UPDATE varia SET v = TO_VARIANT(float1); -- converts FROM a float TO a variant.

UPDATE varia SET float2 = v::FLOAT; -- converts FROM a variant TO a float.

The screenshot shows the Snowflake UI interface. At the top, it displays the connection name "SAMPLE_SNOW"."Snowflake" and various navigation options like "Settings" and "Code Versions". Below the connection bar, two UPDATE statements are shown: "UPDATE varia SET v = TO_VARIANT(float1); -- converts FROM a float TO a variant." and "UPDATE varia SET float2 = v::FLOAT; -- converts FROM a variant TO a float.". In the main area, there's a "Results" tab selected, showing a table with one row labeled "number of rows updated". The value is 1. To the right of the results table, a "Query Details" panel is visible, showing a progress bar for "Query duration" (403ms).

3. Check the Data.



The screenshot shows a Snowflake query results page. The query is:

```
1 | SELECT * FROM varia;
```

The results table has two columns: FLOAT1 and FLOAT2. Both columns contain the value 1.23. The results panel also displays query details: Query duration 136ms.

	FLOAT1	V	FLOAT2
1	1.23	1.23000000000000e+00	1.23

Query Details
Query duration 136ms

4. To convert a value from the VARIANT data type, specify the data type that you want to convert to.
5. For example, the following statement uses the :: operator to specify that the VARIANT should be converted to FLOAT:

Query: `SELECT v::FLOAT * 3.14 as Result FROM varia;`



The screenshot shows a Snowflake query results page. The query is:

```
1 | SELECT v::FLOAT * 3.14 as Result FROM varia;
```

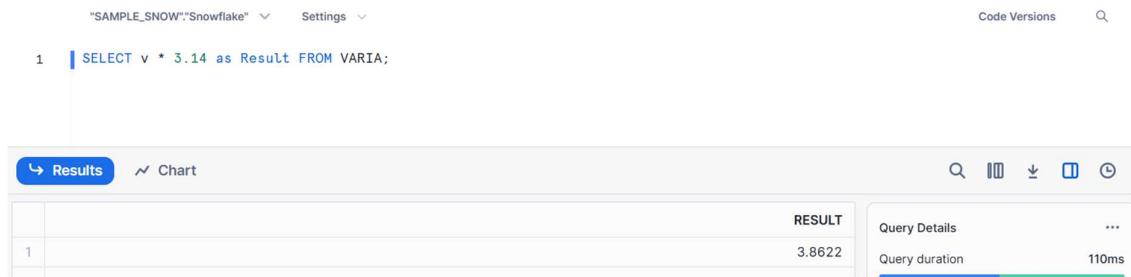
The results table has one column: RESULT, which contains the value 3.8622. The results panel also displays query details: Query duration 170ms.

RESULT
3.8622

Query Details
Query duration 170ms

6. VARIANT stores both the value and the data type of the value.
7. This allows you to also use VARIANT values in expressions where the value's data type is valid without first casting the VARIANT.
8. For example, if VARIANT column my_variant_column contains a numeric value, then you can directly multiply my_variant_column by another numeric value:

Query: `SELECT v * 3.14 as Result FROM VARIA;`



The screenshot shows a Snowflake query results page. The query is:

```
1 | SELECT v * 3.14 as Result FROM VARIA;
```

The results table has one column: RESULT, which contains the value 3.8622. The results panel also displays query details: Query duration 110ms.

RESULT
3.8622

Query Details
Query duration 110ms

Array

1. Create a Simple Array.

Query: `SELECT ARRAY_CONSTRUCT(1, 2, 3, 4, 5) AS simple_array;`

The screenshot shows a Snowflake query editor interface. The top bar includes the connection name "SAMPLE_SNOW"."Snowflake", settings, code versions, and a search icon. The main area contains the following SQL code:

```
1 | SELECT ARRAY_CONSTRUCT(1, 2, 3, 4, 5) AS simple_array;
```

The results pane shows a single row with the column name "SIMPLE_ARRAY" and the value "[1, 2, 3, 4, 5]". On the right side, there are "Query Details" and "Query duration" sections.

2. Creating an Array from a Table Column.

Query: `CREATE OR REPLACE TABLE sample_table (id NUMBER);
INSERT INTO sample_table VALUES (1), (2), (3), (4), (5);

SELECT ARRAY_AGG(id) AS array_from_column
FROM sample_table;`

`SELECT ARRAY_AGG(id) AS array_from_column
FROM sample_table;`

The screenshot shows a Snowflake query editor interface. The top bar includes the connection name "SAMPLE_SNOW"."Snowflake", settings, code versions, and a search icon. The main area contains the following SQL code:

```
1 | CREATE OR REPLACE TABLE sample_table (id NUMBER);  
2 | INSERT INTO sample_table VALUES (1), (2), (3), (4), (5);  
3 |  
4 | SELECT ARRAY_AGG(id) AS array_from_column  
5 | FROM sample_table;
```

The results pane shows a single row with the column name "ARRAY_FROM_COLUMN" and the value "[1, 2, 3, 4, 5]". On the right side, there are "Query Details" and "Query duration" sections.

3. Accessing a Specific Element.

Query: `SELECT simple_array[2] AS second_element
FROM (SELECT ARRAY_CONSTRUCT(1, 2, 3, 4, 5) AS simple_array);`

The screenshot shows a Snowflake query editor interface. The top bar includes the connection name "SAMPLE_SNOW"."Snowflake", settings, code versions, and a search icon. The main area contains the following SQL code:

```
1 | SELECT simple_array[2] AS second_element  
2 | FROM (SELECT ARRAY_CONSTRUCT(1, 2, 3, 4, 5) AS simple_array);
```

The results pane shows a single row with the column name "SECOND_ELEMENT" and the value "3". On the right side, there are "Query Details" and "Query duration" sections.

4. Accessing All Elements Using FLATTEN.

Query: SELECT value AS array_element

```
FROM TABLE(FLATTEN(INPUT => ARRAY_CONSTRUCT(1, 2, 3, 4, 5)));
```

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW"."Snowflake" and "Settings". On the right, there are "Code Versions" and a search icon. Below the navigation is the query editor with the following code:

```
1  SELECT value AS array_element
2  FROM TABLE(FLATTEN(INPUT => ARRAY_CONSTRUCT(1, 2, 3, 4, 5)));
```

Below the editor is the results section, which has tabs for "Results" and "Chart". The "Results" tab is selected, showing a table with one column named "ARRAY_ELEMENT". The data rows are 1, 2, 3, 4, and 5. To the right of the results table is a "Query Details" panel. It shows "Query duration" as 209ms, "Rows" as 5, and "Query ID" as 01b51c58-0001-2586-0... There is also a small "ARRAY_ELEMENT" button with a dropdown menu icon.

5. Appending an Element to an Array.

Query: SELECT ARRAY_APPEND(ARRAY_CONSTRUCT(1, 2, 3), 4) AS updated_array;

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW"."Snowflake" and "Settings". On the right, there are "Code Versions" and a search icon. Below the navigation is the query editor with the following code:

```
1  SELECT ARRAY_APPEND(ARRAY_CONSTRUCT(1, 2, 3), 4) AS updated_array;
```

Below the editor is the results section, which has tabs for "Results" and "Chart". The "Results" tab is selected, showing a table with one column named "UPDATED_ARRAY". The data row is [1, 2, 3, 4]. To the right of the results table is a "Query Details" panel. It shows "Query duration" as 146ms.

6. Prepending an Element to an Array.

Query: SELECT ARRAY-prepend(ARRAY_CONSTRUCT(0), ARRAY_CONSTRUCT(1, 2, 3)) AS updated_array;

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW"."Snowflake" and "Settings". On the right, there are "Code Versions" and a search icon. Below the navigation is the query editor with the following code:

```
1  SELECT ARRAY-prepend(ARRAY_CONSTRUCT(0), ARRAY_CONSTRUCT(1, 2, 3)) AS updated_array;
```

Below the editor is the results section, which has tabs for "Results" and "Chart". The "Results" tab is selected, showing a table with one column named "UPDATED_ARRAY". The data row is [[1, 2, 3], 0]. To the right of the results table is a "Query Details" panel. It shows "Query duration" as 146ms.

Object

1. Creating a Simple Object.

Query: `SELECT OBJECT_CONSTRUCT('name', 'Alice', 'age', 30) AS simple_object;`



The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW"."Snowflake" and "Settings". On the right, there are "Code Versions" and a search icon. Below the navigation is a code editor window containing the query:

```
1 | SELECT OBJECT_CONSTRUCT('name', 'Alice', 'age', 30) AS simple_object;
```

Below the code editor is the results pane, which has tabs for "Results" and "Chart". The "Results" tab is selected. It displays a table with one row labeled "SIMPLE_OBJECT". The data in the table is:

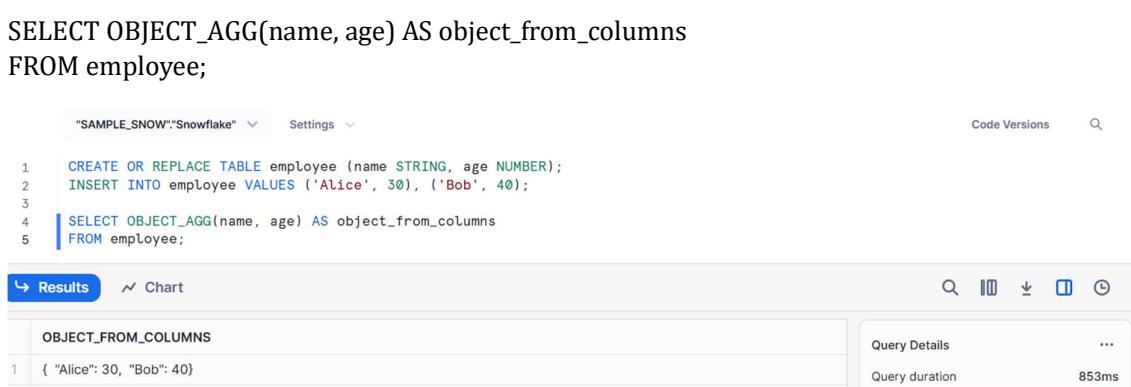
	SIMPLE_OBJECT
1	{ "age": 30, "name": "Alice"}

On the right side of the results pane, there's a "Query Details" panel showing "Query duration" as 174ms. There are also icons for search, refresh, and more.

2. Creating an Object from Table Columns.

Query: `CREATE OR REPLACE TABLE employee (name STRING, age NUMBER);
INSERT INTO employee VALUES ('Alice', 30), ('Bob', 40);

SELECT OBJECT_AGG(name, age) AS object_from_columns
FROM employee;`



The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW"."Snowflake" and "Settings". On the right, there are "Code Versions" and a search icon. Below the navigation is a code editor window containing the query:

```
1 | CREATE OR REPLACE TABLE employee (name STRING, age NUMBER);  
2 | INSERT INTO employee VALUES ('Alice', 30), ('Bob', 40);  
3 |  
4 | SELECT OBJECT_AGG(name, age) AS object_from_columns  
5 | FROM employee;
```

Below the code editor is the results pane, which has tabs for "Results" and "Chart". The "Results" tab is selected. It displays a table with one row labeled "OBJECT_FROM_COLUMNS". The data in the table is:

	OBJECT_FROM_COLUMNS
1	{ "Alice": 30, "Bob": 40}

On the right side of the results pane, there's a "Query Details" panel showing "Query duration" as 853ms. There are also icons for search, refresh, and more.

3. Accessing a Specific Key-Value Pair.

Query: `SELECT simple_object:name AS employee_name
FROM (SELECT OBJECT_CONSTRUCT('name', 'Alice', 'age', 30) AS simple_object);`



The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with "SAMPLE_SNOW"."Snowflake" and "Settings". On the right, there are "Code Versions" and a search icon. Below the navigation is a code editor window containing the query:

```
1 | SELECT simple_object:name AS employee_name  
2 | FROM (SELECT OBJECT_CONSTRUCT('name', 'Alice', 'age', 30) AS simple_object);
```

Below the code editor is the results pane, which has tabs for "Results" and "Chart". The "Results" tab is selected. It displays a table with one row labeled "EMPLOYEE_NAME". The data in the table is:

	EMPLOYEE_NAME
1	"Alice"

On the right side of the results pane, there's a "Query Details" panel showing "Query duration" as 172ms. There are also icons for search, refresh, and more.

4. Accessing All Key-Value Pairs Using FLATTEN.

Query: SELECT key, value

```
FROM TABLE(FLATTEN(INPUT => OBJECT_CONSTRUCT('name', 'Alice', 'age', 30)));
```

The screenshot shows a Snowflake query editor with the following details:

- Connection: "SAMPLE_SNOW"."Snowflake"
- Code Versions: Available
- Results tab selected.
- Query: `SELECT key, value FROM TABLE(FLATTEN(INPUT => OBJECT_CONSTRUCT('name', 'Alice', 'age', 30)));`
- Table Results:

KEY	VALUE
age	30
name	"Alice"
- Query Details:
 - Query duration: 125ms
 - Rows: 2

5. Adding a New Key-Value Pair.

Query: SELECT OBJECT_INSERT(OBJECT_CONSTRUCT('name', 'Alice', 'age', 30), 'department', 'Engineering') AS updated_object;

The screenshot shows a Snowflake query editor with the following details:

- Connection: "SAMPLE_SNOW"."Snowflake"
- Code Versions: Available
- Results tab selected.
- Query: `SELECT OBJECT_INSERT(OBJECT_CONSTRUCT('name', 'Alice', 'age', 30), 'department', 'Engineering') AS updated_object;`
- Table Results:

UPDATED_OBJECT
{ "age": 30, "department": "Engineering", "name": "Alice"}
- Query Details:
 - Query duration: 137ms

6. Removing a Key-Value Pair.

Query: SELECT OBJECT_DELETE(OBJECT_CONSTRUCT('name', 'Alice', 'age', 30), 'age') AS updated_object;

The screenshot shows a Snowflake query editor with the following details:

- Connection: "SAMPLE_SNOW"."Snowflake"
- Code Versions: Available
- Results tab selected.
- Query: `SELECT OBJECT_DELETE(OBJECT_CONSTRUCT('name', 'Alice', 'age', 30), 'age') AS updated_object;`
- Table Results:

UPDATED_OBJECT
{ "name": "Alice"}
- Query Details:
 - Query duration: 178ms

Flatten Array

1. Create a table with the Array field and insert data.

Query: create or replace transient table emp01(

```
    id number,  
    first_name varchar,  
    last_name varchar,  
    designation varchar,  
    certifications array  
);
```

```
insert into emp01
```

```
select 1, 'Alexander', 'Kostas', 'Snowflake Developer', array_construct('SnowPro Core');
```

```
insert into emp01
```

```
select 2, 'Pierre', 'Dupont', 'Sr. Snowflake Developer', array_construct('SnowPro Core', 'SnowPro Adv DE');
```

```
insert into emp01
```

```
select 3, 'Isabella', 'Rossi', 'Snowflake Architect', array_construct('SnowPro Core', 'SnowPro Adv DE', 'SnowPro Architect');
```



```
"SAMPLE_SNOW"."Snowflake" Settings Code Versions   
1 create or replace transient table emp01(  
2     id number,  
3     first_name varchar,  
4     last_name varchar,  
5     designation varchar,  
6     certifications array  
7 );  
8  
9     insert into emp01  
10    select 1, 'Alexander', 'Kostas', 'Snowflake Developer', array_construct('SnowPro Core');  
11    insert into emp01  
12    select 2, 'Pierre', 'Dupont', 'Sr. Snowflake Developer', array_construct('SnowPro Core', 'SnowPro Adv DE');  
13    insert into emp01  
14    select 3, 'Isabella', 'Rossi', 'Snowflake Architect', array_construct('SnowPro Core', 'SnowPro Adv DE', 'SnowPro Architect');
```

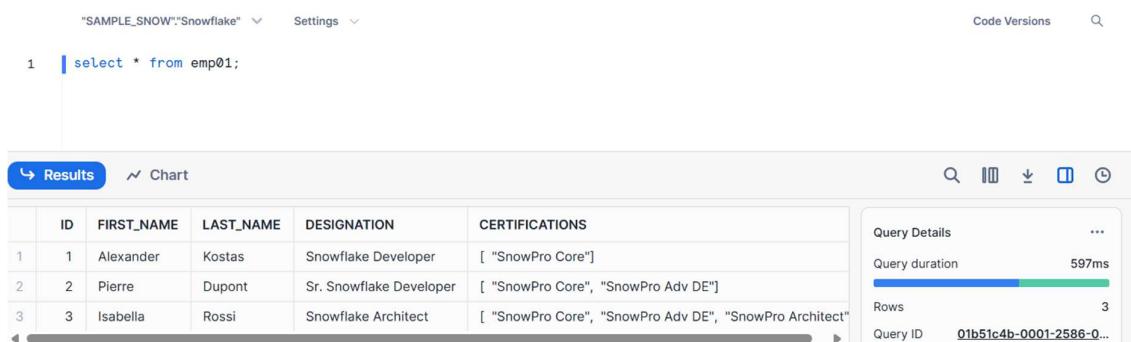
Results Chart

	ID	FIRST_NAME	LAST_NAME	DESIGNATION	CERTIFICATIONS
1	1	Alexander	Kostas	Snowflake Developer	["SnowPro Core"]
2	2	Pierre	Dupont	Sr. Snowflake Developer	["SnowPro Core", "SnowPro Adv DE"]
3	3	Isabella	Rossi	Snowflake Architect	["SnowPro Core", "SnowPro Adv DE", "SnowPro Architect"]

number of rows inserted: 1

Query Details ...
Query duration 349ms

2. Check the data.



```
"SAMPLE_SNOW"."Snowflake" Settings Code Versions   
1 select * from emp01;
```

Results Chart

	ID	FIRST_NAME	LAST_NAME	DESIGNATION	CERTIFICATIONS
1	1	Alexander	Kostas	Snowflake Developer	["SnowPro Core"]
2	2	Pierre	Dupont	Sr. Snowflake Developer	["SnowPro Core", "SnowPro Adv DE"]
3	3	Isabella	Rossi	Snowflake Architect	["SnowPro Core", "SnowPro Adv DE", "SnowPro Architect"]

Query Details ...
Query duration 597ms
Rows 3
Query ID 01b51c4b-0001-2586-0...

3. Flatten an Array and List All Certifications.

Query: select flatten_tbl.value::varchar as certification

```
from table(flatten(input => array_construct('SnowPro Core','SnowPro Adv DE','SnowPro Architect'))) flatten_tbl
```

```
order by 1;
```

```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
```

```
1 | select flatten_tbl.value::varchar as certification
2 | from table(flatten(input => array_construct('SnowPro Core','SnowPro Adv DE','SnowPro Architect'))) flatten_tbl
3 | order by 1;
```

CERTIFICATION
1 SnowPro Adv DE
2 SnowPro Architect
3 SnowPro Core

Query Details

Query duration 52ms

Rows 3

Query ID 01b51c4c-0001-2586-0...

4. This query effectively lists each employee with their certifications in separate rows, making it easier to see the specific certifications each employee holds.

Query: select emp.first_name, emp.last_name, cert.value::varchar as cert_name

```
from emp01 emp,
```

```
lateral flatten(input => emp.certifications) cert;
```

```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
```

```
1 | select emp.first_name, emp.last_name, cert.value::varchar as cert_name
2 | from emp01 emp,
3 | lateral flatten(input => emp.certifications) cert;
```

FIRST_NAME	LAST_NAME	CERT_NAME
1 Alexander	Kostas	SnowPro Core
2 Pierre	Dupont	SnowPro Core
3 Pierre	Dupont	SnowPro Adv DE
4 Isabella	Rossi	SnowPro Core
5 Isabella	Rossi	SnowPro Adv DE
6 Isabella	Rossi	SnowPro Architect

Query Details

Query duration 161ms

Rows 6

Query ID 01b51c4d-0001-2587-0...

FIRST_NAME

Isabella
3

Flatten Object

1. Create a table with the object field and insert data.

```
Query: create or replace transient table emp02(
    id number,
    first_name varchar,
    last_name varchar,
    designation varchar,
    certifications object
);

insert into emp02
select 1, 'Alexander', 'Kostas', 'Snowflake Developer',
       object_construct('SnowPro Core', '790');
insert into emp02
select 2, 'Pierre', 'Dupont', 'Sr. Snowflake Developer',
       object_construct('SnowPro Core', '890', 'SnowPro Adv DE', '810');
insert into emp02
select 3, 'Isabella', 'Rossi', 'Snowflake Architect',
       object_construct('SnowPro Core', '950', 'SnowPro Adv DE', '780', 'SnowPro
Architect', '900');
```

The screenshot shows the Snowflake UI interface. At the top, it displays the connection name "SAMPLE_SNOW" and a "Snowflake" icon. On the right, there are "Code Versions" and a search bar. Below the connection info, the SQL query is displayed in a code editor. The code creates a transient table emp02 with columns id, first_name, last_name, designation, and certifications. It then inserts three rows into the table. The first row has id 1, first_name 'Alexander', last_name 'Kostas', designation 'Snowflake Developer', and certifications object 'SnowPro Core' with value '790'. The second row has id 2, first_name 'Pierre', last_name 'Dupont', designation 'Sr. Snowflake Developer', and certifications object 'SnowPro Core' with value '890', and another object 'SnowPro Adv DE' with value '810'. The third row has id 3, first_name 'Isabella', last_name 'Rossi', designation 'Snowflake Architect', and certifications object 'SnowPro Core' with value '950', another object 'SnowPro Adv DE' with value '780', and another object 'SnowPro Architect' with value '900'. After running the query, the results tab shows a single row inserted. On the right, there is a "Query Details" panel showing a blue progress bar for "Query duration" and a green bar for "Rows" (3). A "Query ID" is also visible.

2. Check the data.

The screenshot shows the Snowflake UI interface. At the top, it displays the connection name "SAMPLE_SNOW" and a "Snowflake" icon. On the right, there are "Code Versions" and a search bar. Below the connection info, the SQL query is displayed in a code editor. The query selects all columns from the emp02 table. The results tab shows the data from the emp02 table. There are three rows: Row 1 has ID 1, FIRST_NAME 'Alexander', LAST_NAME 'Kostas', DESIGNATION 'Snowflake Developer', and CERTIFICATIONS object with key 'SnowPro Core' and value '790'. Row 2 has ID 2, FIRST_NAME 'Pierre', LAST_NAME 'Dupont', DESIGNATION 'Sr. Snowflake Developer', and CERTIFICATIONS object with keys 'SnowPro Core' and 'SnowPro Adv DE' and values '890' and '810'. Row 3 has ID 3, FIRST_NAME 'Isabella', LAST_NAME 'Rossi', DESIGNATION 'Snowflake Architect', and CERTIFICATIONS object with keys 'SnowPro Core', 'SnowPro Adv DE', and 'SnowPro Architect' and values '950', '780', and '900'. On the right, there is a "Query Details" panel showing a blue progress bar for "Query duration" (204ms) and a green bar for "Rows" (3). A "Query ID" is also visible.

3. Create an array with three certification names and then flattens the array, outputting each certification name as a separate row.
4. The ::varchar cast ensures the values are treated as strings. The result is ordered alphabetically.

Query: `SELECT flatten_tbl.value::varchar AS certification FROM TABLE(FLATTEN (input => ARRAY_CONSTRUCT('SnowPro Core','SnowPro Adv DE','SnowPro Architect'))) flatten_tbl ORDER BY 1;`

```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
Code Versions ⚡
1 | SELECT flatten_tbl.value::varchar AS certification FROM TABLE(FLATTEN (input => ARRAY_CONSTRUCT('SnowPro Core','SnowPro Adv DE','SnowPro Architect'))) flatten_tbl
2 | ORDER BY 1;

Results ▾ Chart
CERTIFICATION
1 SnowPro Adv DE
2 SnowPro Architect
3 SnowPro Core

Query Details ...
Query duration 135ms
Rows 3
Query ID 01b51c51-0001-2587-0...
```

5. This below query joins each employee's record with the flattened certifications object.
6. The LATERAL FLATTEN function is used to unnest the JSON object, treating each key-value pair as a separate row.
7. The output shows each employee's first name, last name, and the names of their certifications.

Query: `SELECT emp.first_name, emp.last_name, cert.value::varchar AS cert_name FROM emp02 emp, LATERAL FLATTEN (input => emp.certifications) cert;`

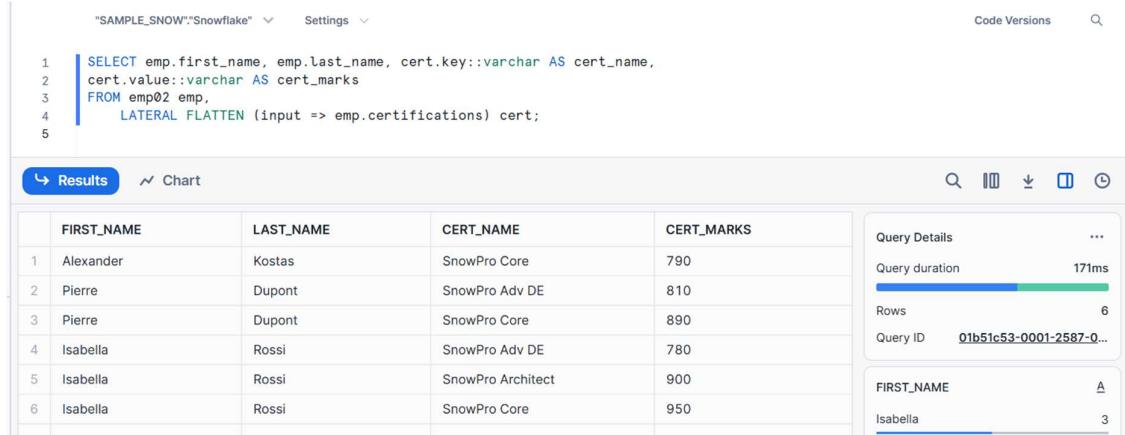
```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
Code Versions ⚡
1 | SELECT emp.first_name, emp.last_name, cert.value::varchar AS cert_name
2 | FROM emp02 emp,
3 |     LATERAL FLATTEN (input => emp.certifications) cert;

Results ▾ Chart
FIRST_NAME LAST_NAME CERT_NAME
1 Alexander Kostas 790
2 Pierre Dupont 810
3 Pierre Dupont 890
4 Isabella Rossi 780
5 Isabella Rossi 900
6 Isabella Rossi 950

Query Details ...
Query duration 162ms
Rows 6
Query ID 01b51c52-0001-2586-0...
```

8. The below query is similar to the previous one but includes both the certification names and their corresponding scores.
9. The cert. key retrieves the certification name, and the cert. value retrieves the score, both cast as strings.

Query: SELECT emp.first_name, emp.last_name, cert.key::varchar AS cert_name, cert.value::varchar AS cert_marks
 FROM emp02 emp,
 LATERAL FLATTEN (input => emp.certifications) cert;



The screenshot shows the Snowflake UI interface. At the top, it displays the connection information: "SAMPLE_SNOW"."Snowflake". Below the code editor, the results are displayed in a table format. The table has four columns: FIRST_NAME, LAST_NAME, CERT_NAME, and CERT_MARKS. The data is as follows:

	FIRST_NAME	LAST_NAME	CERT_NAME	CERT_MARKS
1	Alexander	Kostas	SnowPro Core	790
2	Pierre	Dupont	SnowPro Adv DE	810
3	Pierre	Dupont	SnowPro Core	890
4	Isabella	Rossi	SnowPro Adv DE	780
5	Isabella	Rossi	SnowPro Architect	900
6	Isabella	Rossi	SnowPro Core	950

On the right side of the results table, there is a sidebar with "Query Details" showing metrics like Query duration (171ms), Rows (6), and Query ID (01b51c53-0001-2587-0...). There is also a "FIRST_NAME" histogram with Isabella having a count of 3.

Extracting Values

1. Create a table named car_sales that contains a single VARIANT column named src. This VARIANT contains nested ARRAYS and OBJECTs.

Query: CREATE OR REPLACE TABLE car_sales

```
{
  src variant
}
AS
SELECT PARSE_JSON(column1) AS src
FROM VALUES
'{
  "date" : "2017-04-28",
  "dealership" : "Valley View Auto Sales",
  "salesperson" : {
    "id": "55",
    "name": "Frank Beasley"
  },
  "customer" : [
    {"name": "Joyce Ridgely", "phone": "16504378889", "address": "San Francisco, CA"}
  ],
  "vehicle" : [
    {"make": "Honda", "model": "Civic", "year": "2017", "price": "20275", "extras":["ext
warranty", "paint protection"]}
  ]
}',
```

```
{
  "date": "2017-04-28",
  "dealership": "Tindel Toyota",
  "salesperson": {
    "id": "274",
    "name": "Greg Northrup"
  },
  "customer": [
    {"name": "Bradley Greenbloom", "phone": "12127593751", "address": "New York, NY"}
  ],
  "vehicle": [
    {"make": "Toyota", "model": "Camry", "year": "2017", "price": "23500", "extras": ["ext warranty", "rust proofing", "fabric protection"]}
  ]
}) v;
```

The screenshot shows the Snowflake UI interface. In the top navigation bar, it says "SAMPLE_SNOW"."Snowflake". On the right, there are "Code Versions" and a search icon. Below the navigation, the code for creating the table is displayed:

```

1 CREATE OR REPLACE TABLE car_sales
2 (
3   src variant
4 )
5 AS
6 SELECT PARSE_JSON(column1) AS src
7 FROM VALUES
8 ('{
9   "date" : "2017-04-28",
10  "dealership" : "Valley View Auto Sales",
11  "salesperson" : {
12    "id": "55",
13    "name": "Frank Beasley"
14  },
15  "customer" : [
16    {"name": "Joyce Ridgely", "phone": "16504378889", "address": "San Francisco, CA"}
17  ],
18  "vehicle" : [
19    {"make": "Honda", "model": "Civic", "year": "2017", "price": "20275", "extras": ["ext warranty", "paint protection"]}
20 ]})

```

Below the code, there are tabs for "Results" (which is selected) and "Chart". The results section shows a single row of data under the "status" column:

status
1 Table CAR_SALES successfully created.

On the right side of the results table, there are "Query Details" and "Query duration" sections. The "Query duration" is listed as 1.9s.

2. Check the Data.

The screenshot shows the Snowflake UI interface. In the top navigation bar, it says "SAMPLE_SNOW"."Snowflake". On the right, there are "Code Versions" and a search icon. Below the navigation, the query is displayed:

```

1 | SELECT * FROM car_sales;

```

Below the query, there are tabs for "Results" (selected) and "Chart". The results section shows two rows of data under the "SRC" column:

SRC
{ "customer": [{ "address": "San Francisco, CA", "name": "Joyce Ridgely", "phone": "16504378889" }], "date": "2017-04-28" }
{ "customer": [{ "address": "New York, NY", "name": "Bradley Greenbloom", "phone": "12127593751" }], "date": "2017-04-28" }

On the right side of the results table, there are "Query Details" and "Query duration" sections. The "Query duration" is listed as 180ms. There are also "Rows" (2) and "Query ID" (01b51c62-0001-2587-0...) details.

3. Get a list of all dealership names by using below query.

Query: `SELECT src:salesperson.name FROM car_sales
ORDER BY 1;`

The screenshot shows a Snowflake query editor interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is the SQL code:

```
1 | SELECT src:salesperson.name FROM car_sales
2 | ORDER BY 1;
```

On the right side, there's a "Results" tab selected, showing the output:

SRC:SALESPERSON.NAME
1 "Frank Beasley"
2 "Greg Northrup"

Below the results, there's a "Query Details" section with a progress bar and the following information:

- Query duration: 279ms
- Rows: 2

4. Get the names of all salespeople who sold cars.

Query: `SELECT src['salesperson']['name'] FROM car_sales
ORDER BY 1;`

The screenshot shows a Snowflake query editor interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is the SQL code:

```
1 | SELECT src['salesperson']['name'] FROM car_sales
2 | ORDER BY 1;
```

On the right side, there's a "Results" tab selected, showing the output:

SRC['SALESPERSON']['NAME']
1 "Frank Beasley"
2 "Greg Northrup"

Below the results, there's a "Query Details" section with a progress bar and the following information:

- Query duration: 194ms
- Rows: 2

5. Retrieve a specific numbered instance of a child element in a repeating array by adding a numbered predicate (starting from 0) to the array reference.
6. Get the vehicle details for each sale

Query: `SELECT src:customer[0].name, src:vehicle[0] FROM car_sales
ORDER BY 1;`

The screenshot shows a Snowflake query editor interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is the SQL code:

```
1 | SELECT src:customer[0].name, src:vehicle[0] FROM car_sales
2 | ORDER BY 1;
```

On the right side, there's a "Results" tab selected, showing the output:

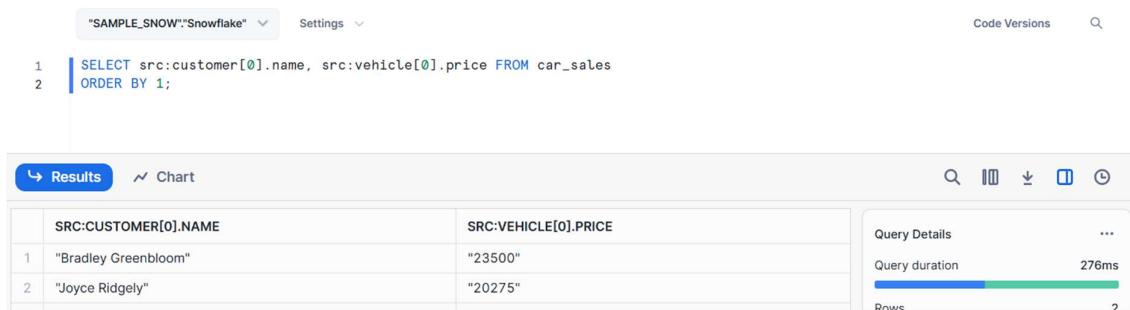
SRC:CUSTOMER[0].NAME	SRC:VEHICLE[0]
1 "Bradley Greenbloom"	{ "extras": ["ext warranty", "rust proofing", "fabric protection"], "make": "Toyota", "model": "RAV4", "year": 2018 }
2 "Joyce Ridgely"	{ "extras": ["ext warranty", "paint protection"], "make": "Honda", "model": "Civic", "year": 2019 }

Below the results, there's a "Query Details" section with a progress bar and the following information:

- Query duration: 21ms
- Rows: 2

- Get the price of each car sold.

Query: `SELECT src:customer[0].name, src:vehicle[0].price FROM car_sales ORDER BY 1;`



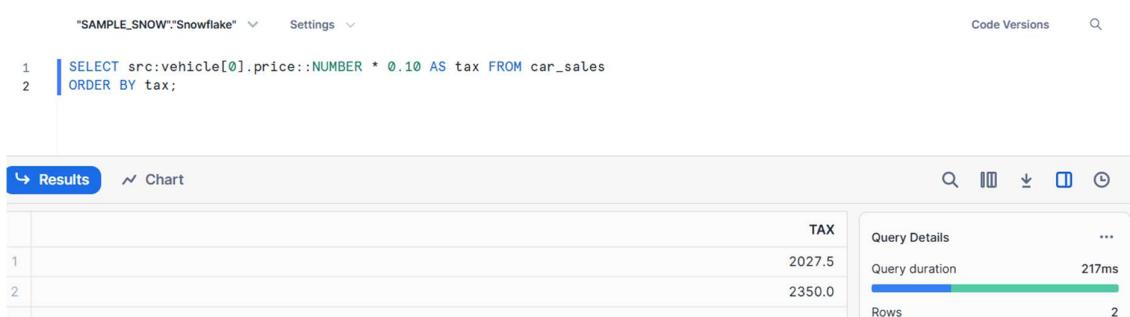
```
"SAMPLE_SNOW"."Snowflake" Settings Code Versions
1 | SELECT src:customer[0].name, src:vehicle[0].price FROM car_sales
2 | ORDER BY 1;

Results Chart
SRC:CUSTOMER[0].NAME SRC:VEHICLE[0].PRICE
1 "Bradley Greenbloom" "23500"
2 "Joyce Ridgely" "20275"

Query Details ...
Query duration 276ms
Rows 2
```

- When you extract values from a VARIANT, you can explicitly cast the values to the desired data type.
- For example, you can extract the prices as numeric values and perform calculations on them.

Query: `SELECT src:vehicle[0].price::NUMBER * 0.10 AS tax FROM car_sales ORDER BY tax;`



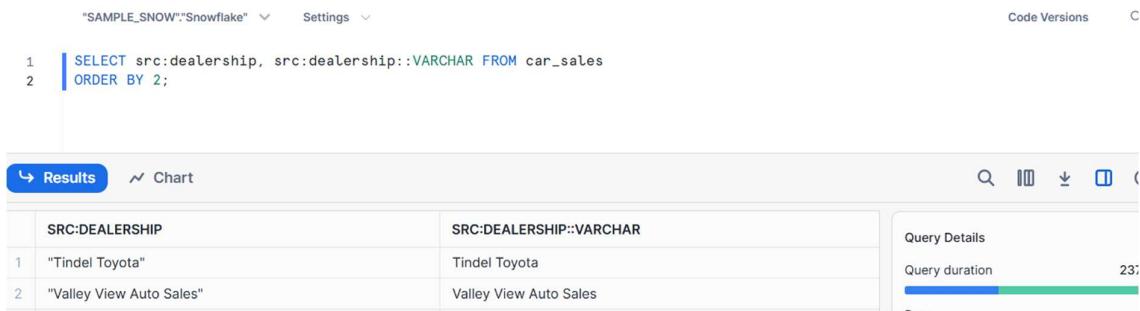
```
"SAMPLE_SNOW"."Snowflake" Settings Code Versions
1 | SELECT src:vehicle[0].price::NUMBER * 0.10 AS tax FROM car_sales
2 | ORDER BY tax;

Results Chart
TAX
1 2027.5
2 2350.0

Query Details ...
Query duration 217ms
Rows 2
```

- By default, when VARCHARs, DATES, TIMES, and TIMESTAMPs are retrieved from a VARIANT column, the values are surrounded by double quotes.
- You can eliminate the double quotes by explicitly casting the values.

Query: `SELECT src:dealership, src:dealership::VARCHAR FROM car_sales ORDER BY 2;`



```
"SAMPLE_SNOW"."Snowflake" Settings Code Versions
1 | SELECT src:dealership, src:dealership::VARCHAR FROM car_sales
2 | ORDER BY 2;

Results Chart
SRC:DEALERSHIP SRC:DEALERSHIP::VARCHAR
1 "Tindel Toyota" Tindel Toyota
2 "Valley View Auto Sales" Valley View Auto Sales

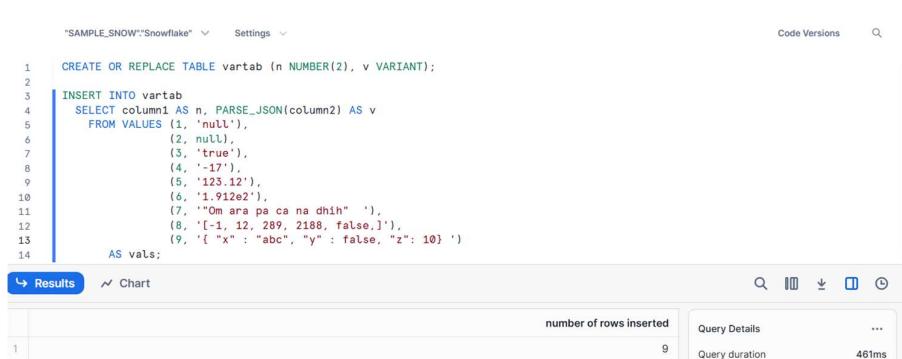
Query Details ...
Query duration 23ms
Rows 2
```

Type Predicates

1. Create a table that contains different types of data stored inside a VARIANT column, then use the TYPEOF function to determine the data type of each one.

Query: CREATE OR REPLACE TABLE vartab (n NUMBER(2), v VARIANT);

```
INSERT INTO vartab
SELECT column1 AS n, PARSE_JSON(column2) AS v
FROM VALUES (1, 'null'),
            (2, null),
            (3, 'true'),
            (4, '-17'),
            (5, '123.12'),
            (6, '1.912e2'),
            (7, '"Om ara pa ca na dhih" '),
            (8, '[-1, 12, 289, 2188, false,]'),
            (9, '{ "x" : "abc", "y" : false, "z": 10 } ')
AS vals;
```

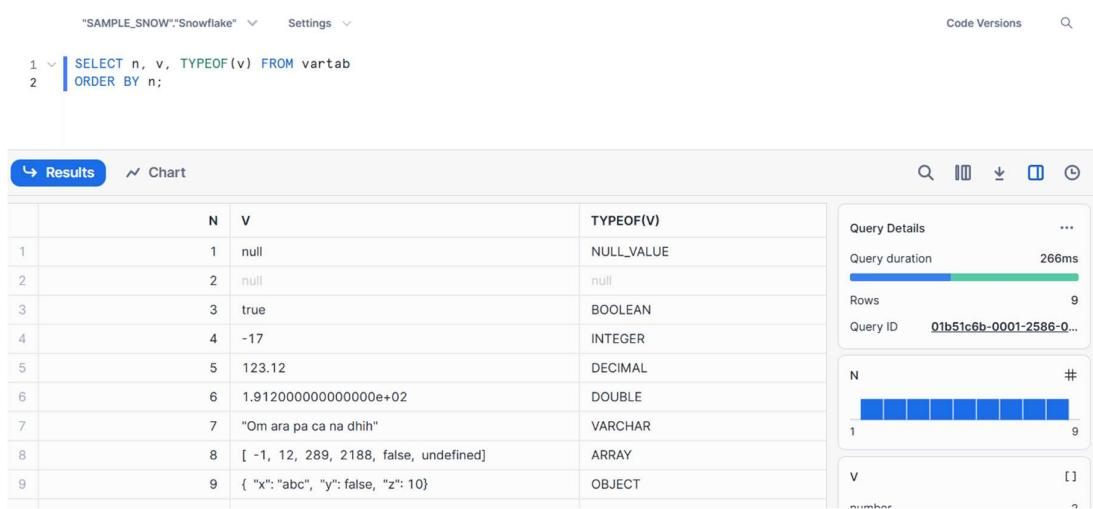


```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
Code Versions
1 CREATE OR REPLACE TABLE vartab (n NUMBER(2), v VARIANT);
2
3 INSERT INTO vartab
4     SELECT column1 AS n, PARSE_JSON(column2) AS v
5         FROM VALUES (1, 'null'),
6                     (2, null),
7                     (3, 'true'),
8                     (4, '-17'),
9                     (5, '123.12'),
10                    (6, '1.912e2'),
11                    (7, '"Om ara pa ca na dhih" '),
12                    (8, '[-1, 12, 289, 2188, false,]'),
13                    (9, '{ "x" : "abc", "y" : false, "z": 10 } ')
14    AS vals;
15
16 Results ▾ Chart
17
18 number of rows inserted 9
19 Query Details ...
20 Query duration 461ms
```

2. Query the Data.

Query: SELECT n, v, TYPEOF(v) FROM vartab

ORDER BY n;



```
"SAMPLE_SNOW"."Snowflake" ▾ Settings ▾
Code Versions
1 SELECT n, v, TYPEOF(v) FROM vartab
2 ORDER BY n;
3
4 Results ▾ Chart
5
6 N V TYPEOF(V)
7 1 null NULL_VALUE
8 2 null null
9 3 true BOOLEAN
10 4 -17 INTEGER
11 5 123.12 DECIMAL
12 6 1.912000000000000e+02 DOUBLE
13 7 "Om ara pa ca na dhih" VARCHAR
14 8 [-1, 12, 289, 2188, false, undefined] ARRAY
15 9 { "x": "abc", "y": false, "z": 10 } OBJECT
16
17 Query Details ...
18 Query duration 266ms
19 Rows 9
20 Query ID 01b51c6b-0001-2586-0...
21
22 N #
23 1 9
24
25 V []
26 number
```

3. The following example uses the TYPEOF function to determine the data type of a value by casting the value to a VARIANT.
4. Create and populate a table.

Query: CREATE OR REPLACE TABLE typeof_cast(status VARCHAR, time TIMESTAMP);

INSERT INTO typeof_cast VALUES('check in', '2024-01-17 19:00:00.000 -0800');

The screenshot shows the Snowflake UI interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is a code editor with three lines of SQL:

```

1 CREATE OR REPLACE TABLE typeof_cast(status VARCHAR, time TIMESTAMP);
2
3 INSERT INTO typeof_cast VALUES('check in', '2024-01-17 19:00:00.000 -0800');

```

Below the code editor is a results table with one row. The table has two columns: "number of rows inserted" and "Query Details".

number of rows inserted	Query Details
1	Query duration 389ms

5. Query the table using the TYPEOF function by casting each value to a VARIANT.

Query: SELECT status, TYPEOF(status::VARIANT) AS "TYPE OF STATUS",
time, TYPEOF(time::VARIANT) AS "TYPE OF TIME"
FROM typeof_cast;

The screenshot shows the Snowflake UI interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is a code editor with three lines of SQL:

```

1 SELECT status, TYPEOF(status::VARIANT) AS "TYPE OF STATUS",
2       time, TYPEOF(time::VARIANT) AS "TYPE OF TIME"
3   FROM typeof_cast;

```

Below the code editor is a results table with four columns: STATUS, TYPE OF STATUS, TIME, and TYPE OF TIME. The table has one row.

STATUS	TYPE OF STATUS	TIME	TYPE OF TIME
check in	VARCHAR	2024-01-17 19:00:00.000	TIMESTAMP_NTZ

On the right side, there is a "Query Details" section showing a progress bar for "Query duration" which is 185ms.

File Functions

GET_STAGE_LOCATION

1. Retrieves the URL for an external or internal named stage using the stage name as the input.
2. Retrieve the URL for an external stage.

Query: CREATE STAGE images_stage URL = 's3://photos/national_parks/us/yosemite/';

SELECT GET_STAGE_LOCATION(@images_stage);

The screenshot shows the Snowflake UI interface. At the top, it says "SAMPLE_SNOW"."Snowflake". Below that is a code editor with three lines of SQL:

```

1 CREATE STAGE images_stage URL = 's3://photos/national_parks/us/yosemite/';
2
3 SELECT GET_STAGE_LOCATION(@images_stage);

```

Below the code editor is a results table with one column: "GET_STAGE_LOCATION(@IMAGES_STAGE)". The table has one row containing the URL "s3://photos/national_parks/us/yosemite/".

GET_STAGE_LOCATION(@IMAGES_STAGE)
s3://photos/national_parks/us/yosemite/

On the right side, there is a "Query Details" section showing a progress bar for "Query duration" which is 21ms.

GET_RELATIVE_PATH

3. Extracts the path of a staged file relative to its location in the stage using the stage name and absolute file path in cloud storage as inputs.
4. Retrieve the relative path of a bitmap format image file in an external stage, where the @images_stage stage references the s3://photos/national_parks/ bucket and path.

Query: SELECT GET_RELATIVE_PATH(@images_stage, 's3://photos/national_parks/us/yosemite/half_dome.jpg');

The screenshot shows the Snowflake UI with the query:

```
1 | SELECT GET_RELATIVE_PATH(@images_stage, 's3://photos/national_parks/us/yosemite/half_dome.jpg');
```

The results pane shows:

GET_RELATIVE_PATH(@IMAGES_STAGE, 'S3://PHOTOS/NATIONAL_PARKS/US/YOSEMITE/HALF_DOME.JPG')
1 half_dome.jpg

Query Details: Query duration 460ms

GET_ABSOLUTE_PATH

5. Retrieves the absolute path of a staged file using the stage name and path of the file relative to its location in the stage as inputs.
6. Retrieve the absolute path of a bitmap format image file in an external stage.

Query: SELECT GET_ABSOLUTE_PATH(@images_stage, 'us/yosemite/half_dome.jpg');

The screenshot shows the Snowflake UI with the query:

```
1 | SELECT GET_ABSOLUTE_PATH(@images_stage, 'us/yosemite/half_dome.jpg');
```

The results pane shows:

GET_ABSOLUTE_PATH(@IMAGES_STAGE, 'US/YOSEMITE/HALF_DOME.JPG')
1 s3://photos/national_parks/us/yosemite/us/yosemite/half_dome.jpg

Query Details: Query duration 57ms

BUILD_SCOPED_FILE_URL

7. Generates a scoped Snowflake file URL to a staged file using the stage name and relative file path as inputs.

Query: SELECT BUILD_SCOPED_FILE_URL(@images_stage,'/us/yosemite/half_dome.jpg');

The screenshot shows the Snowflake UI with the query:

```
1 | SELECT BUILD_SCOPED_FILE_URL(@images_stage, '/us/yosemite/half_dome.jpg');
```

The results pane shows:

BUILD_SCOPED_FILE_URL(@IMAGES_STAGE, '/US/YOSEMITE/HALF_DOME.JPG')
1 https://hr09072.central-india.azure.snowflakecomputing.com/api/files/01b51c9e-0001-2587-0000-0004e88e7005/210815095

Query Details: Query duration 45ms

BUILD_STAGE_FILE_URL

8. Generates a Snowflake file URL to a staged file using the stage name and relative file path as inputs. A file URL permits prolonged access to a specified file.
9. That is, the file URL does not expire.
10. Retrieve a file URL for a bitmap format image file in an external stage.

Query: SELECT BUILD_STAGE_FILE_URL(@images_stage,'/us/yosemite/half_dome.jpg');



The screenshot shows the Snowflake UI interface. At the top, there is a navigation bar with a dropdown for "SAMPLE_SNOW", a "Snowflake" button, "Settings", "Code Versions", and a search icon. Below the navigation bar, the query editor contains the following SQL code:

```
1 | SELECT BUILD_STAGE_FILE_URL(@images_stage,'/us/yosemite/half_dome.jpg');
```

Below the query editor is the results section. It has tabs for "Results" (which is selected) and "Chart". The results table contains one row:

	BUILD_STAGE_FILE_URL(@IMAGES_STAGE,/US/YOSEMITE/HALF_DOME.JPG)
1	https://hr09072.central-india.azure.snowflakecomputing.com/api/files/SAMPLE_SNOW%22Snowflake%22/IMAGES_STAGE/%2fus

On the right side of the results table, there is a "Query Details" panel. It shows the following information:

- Query duration: 25ms