Context

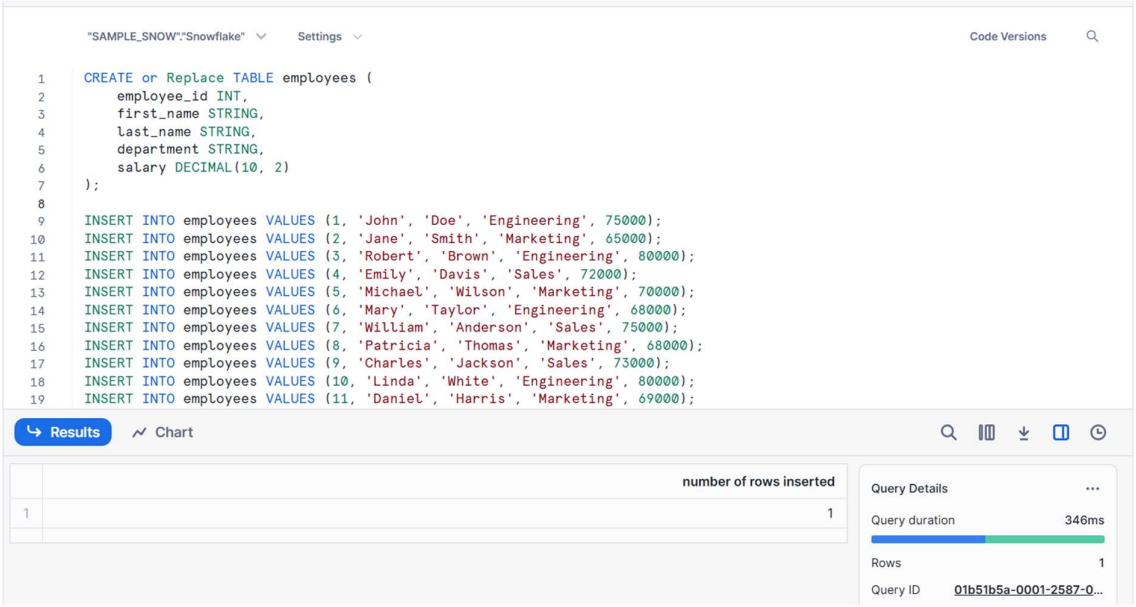
SAMPLE command

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);

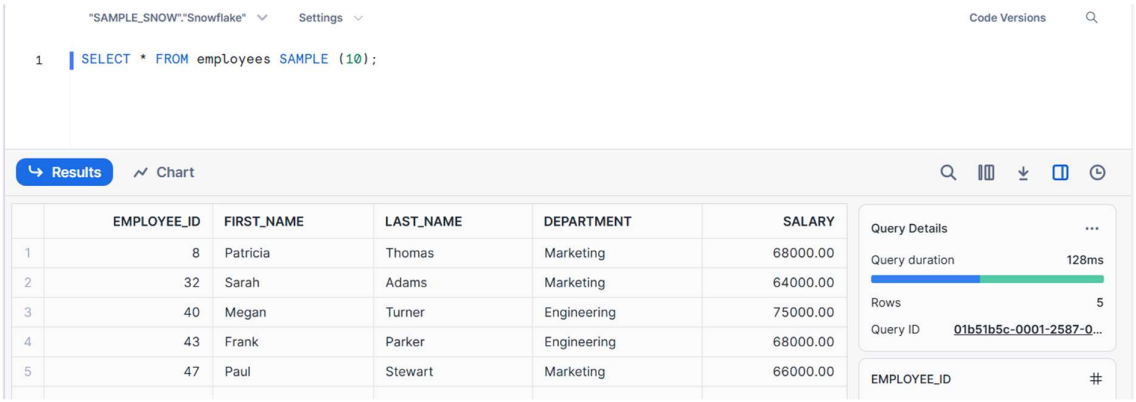
```
INSERT INTO employees VALUES (39, 'Scott', 'Roberts', 'Sales', 71000);
INSERT INTO employees VALUES (40, 'Megan', 'Turner', 'Engineering', 75000);
INSERT INTO employees VALUES (41, 'Richard', 'Phillips', 'Marketing', 64000);
INSERT INTO employees VALUES (42, 'Patricia', 'Campbell', 'Sales', 72000);
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);
```



2. Fraction-based sampling (10%)

Query: SELECT * FROM employees SAMPLE (10);

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



TABLESAMPLE Command

1. Fixed-size sampling (5 rows).

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

2. Exactly 5 rows selected randomly using Bernoulli sampling.

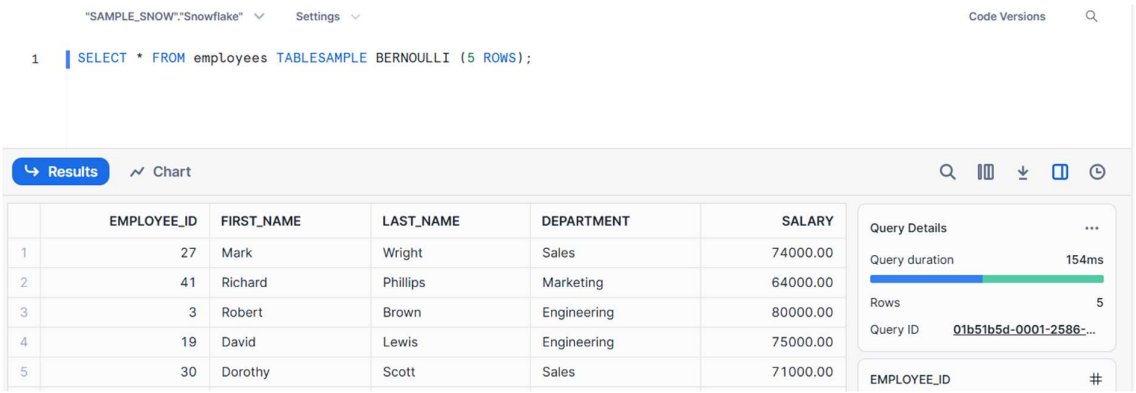
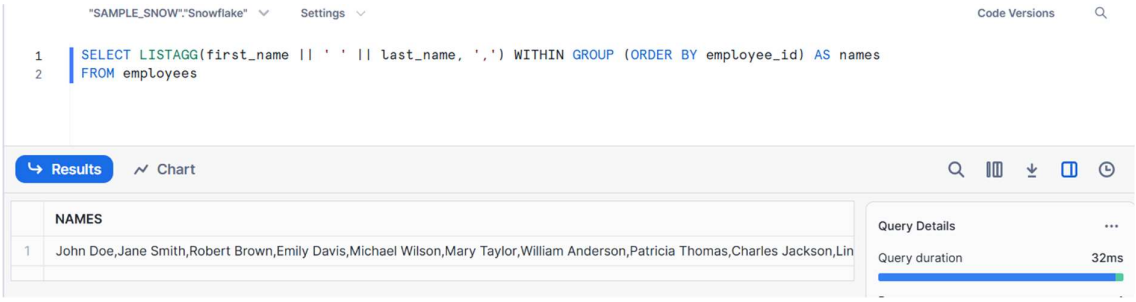


Table Functions

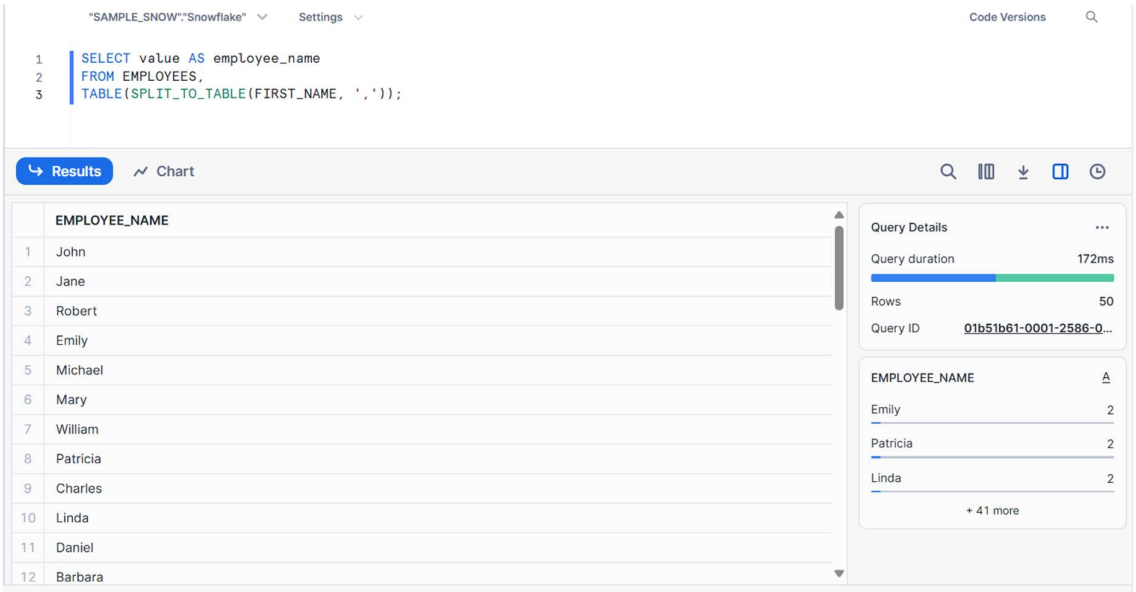
1. The SPLIT_TO_TABLE function splits a string into a table of rows. We'll use this function to split a comma-separated string.
2. Creating a comma-separated string of employee names.

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



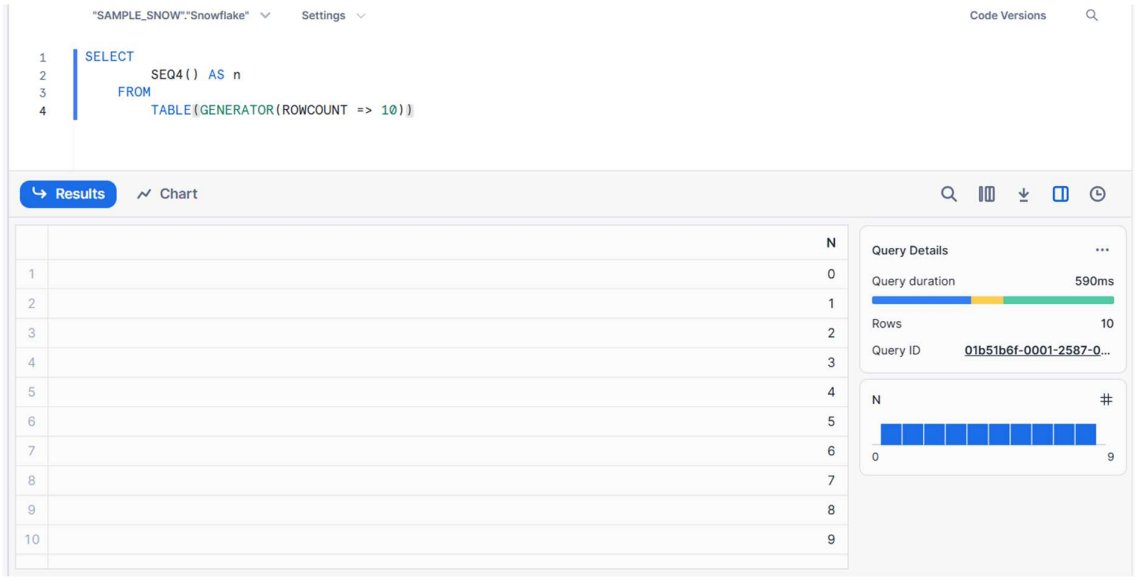
3. Using SPLIT_TO_TABLE to split the string into separate rows.

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



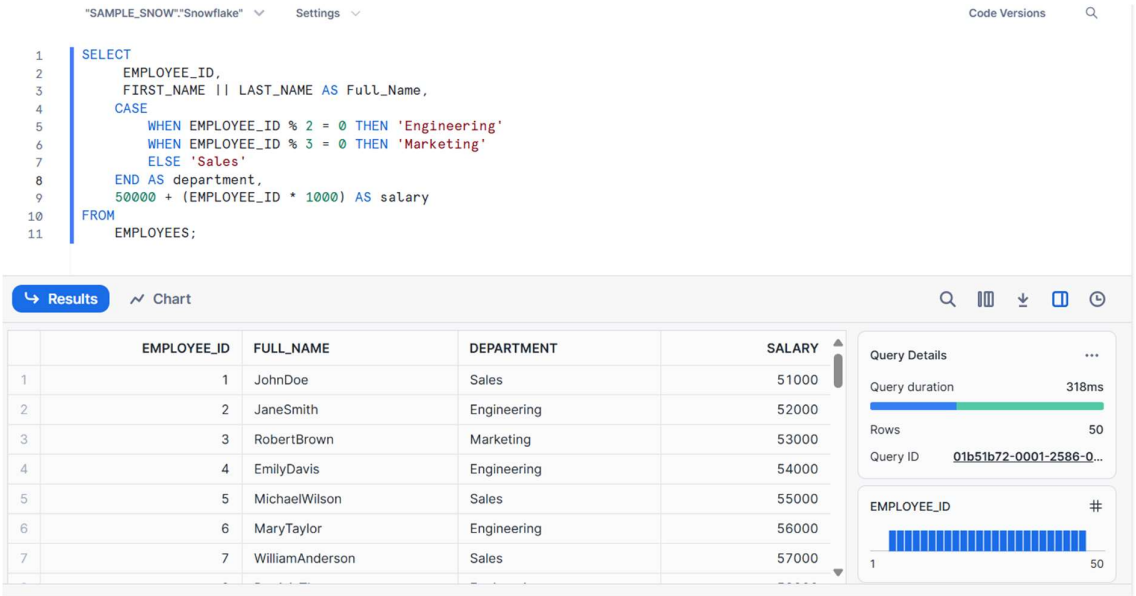
4. Using GENERATOR to create a series of numbers.

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



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;

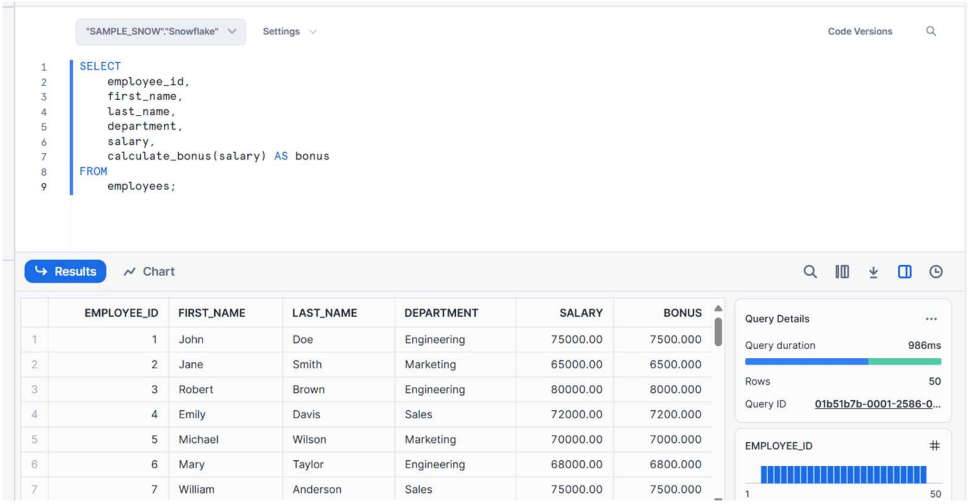


4. Creating a User-Defined Function (UDF) and using it in conjunction with a table function.
5. Create a UDF to calculate a bonus for each employee.

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

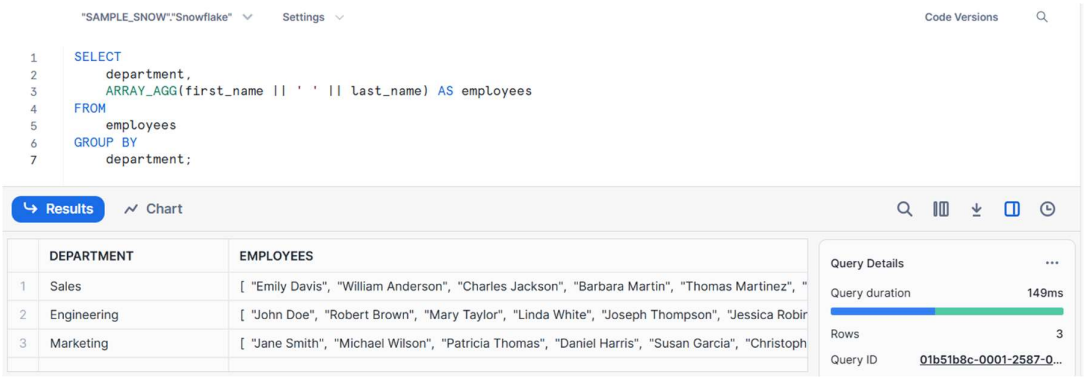
6. Use this UDF in a query to display the bonus along with employee details.

Query: SELECT
  employee_id,
  first_name,
  last_name,
  department,
  salary,
  calculate_bonus(salary) AS bonus
FROM
  employees;



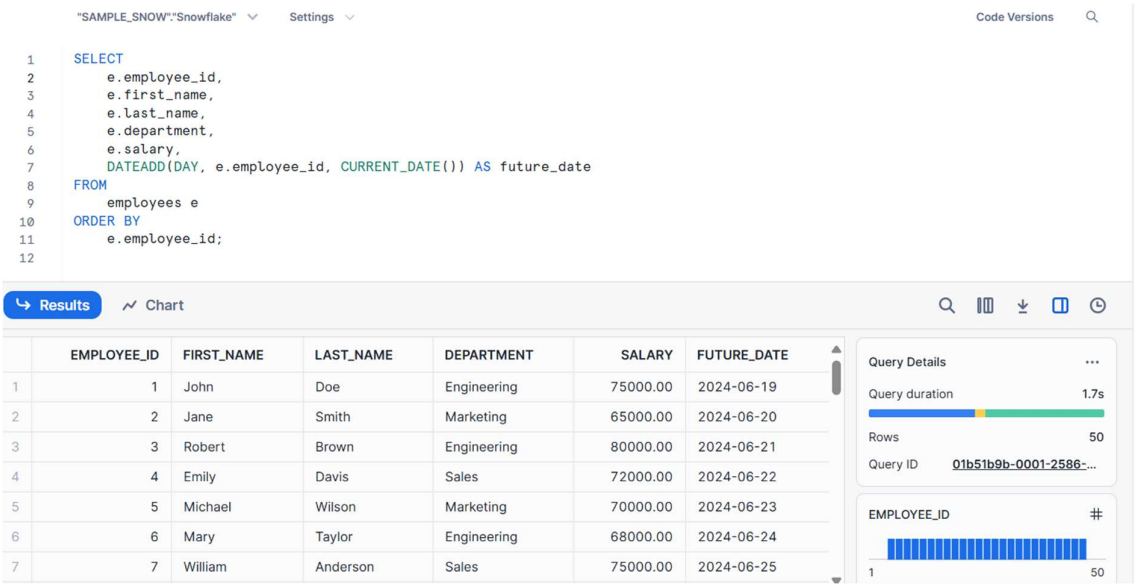
7. The ARRAY_AGG function aggregates rows into an array.
8. Aggregate employee names into an array based on their department.
9. Use ARRAY_AGG with GROUP BY to create an array of employee names for each department.

Query: SELECT
  department,
  ARRAY_AGG(first_name || ' ' || last_name) AS employees
FROM
  employees
GROUP BY
  department;



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;



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.

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.

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.

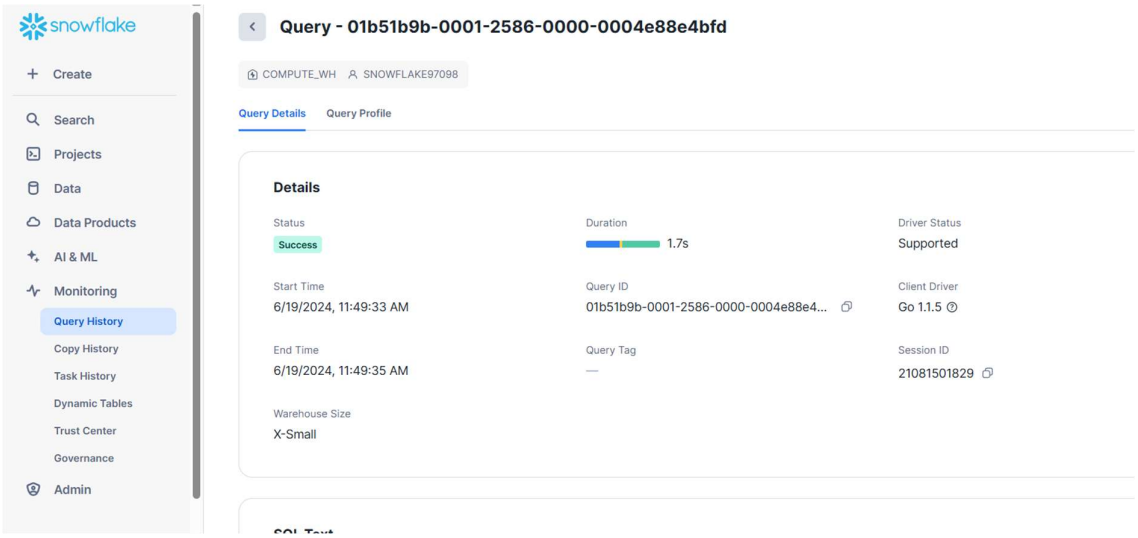
9. Initialize HLL and estimate unique employee IDs.

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.

System Functions

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



2. Cancel the query that you specified by id.

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

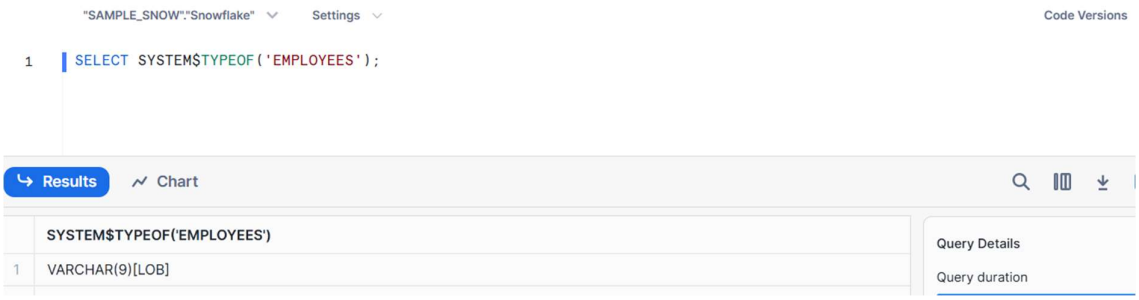
3. Right now we are not executing this query. It will show Identified SQL statement is not currently executing.

4. Abort session that you specified by id.

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

5. Get the type of the object.

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



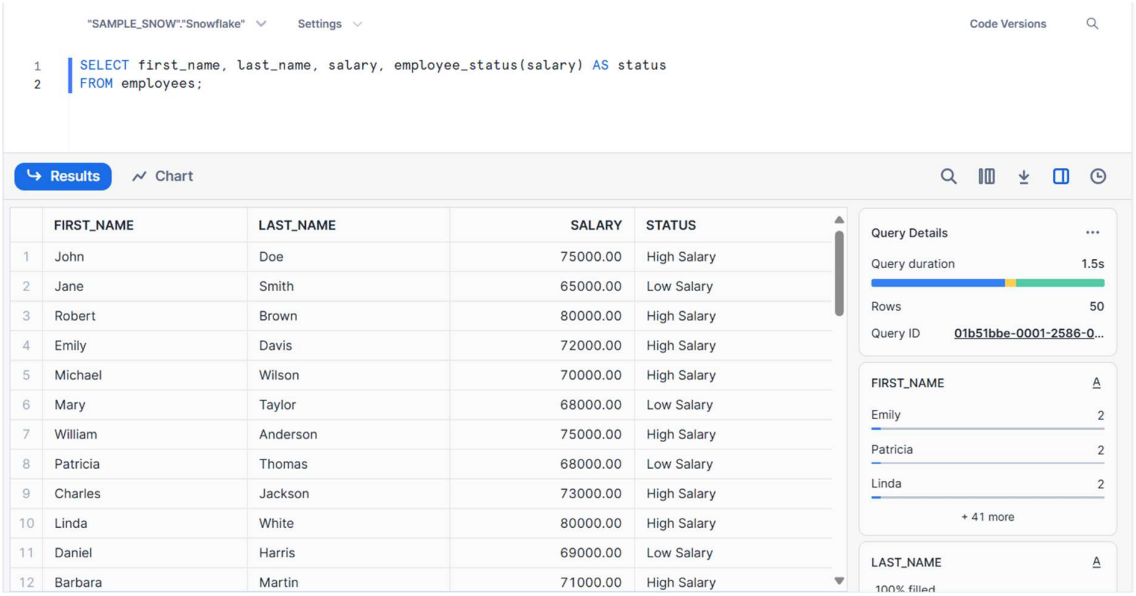
User Defined Functions

1. 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
';

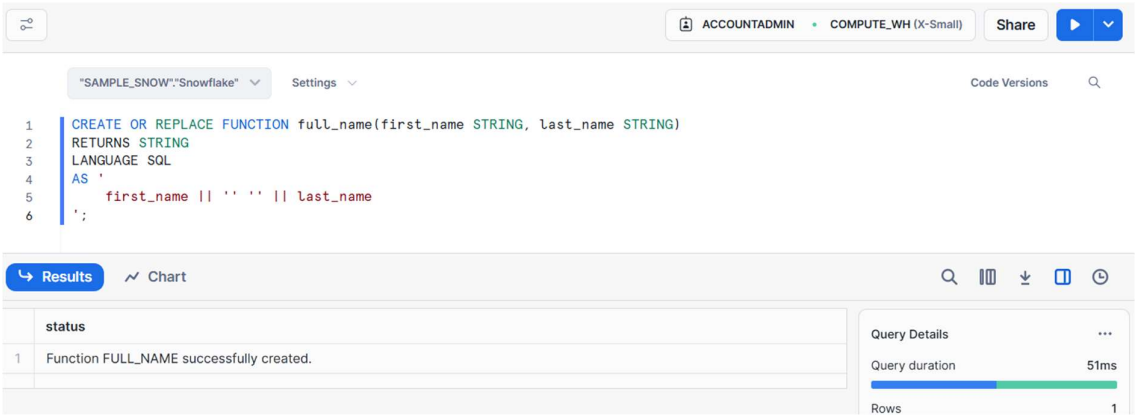
2. Determine employee status based on salary.

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



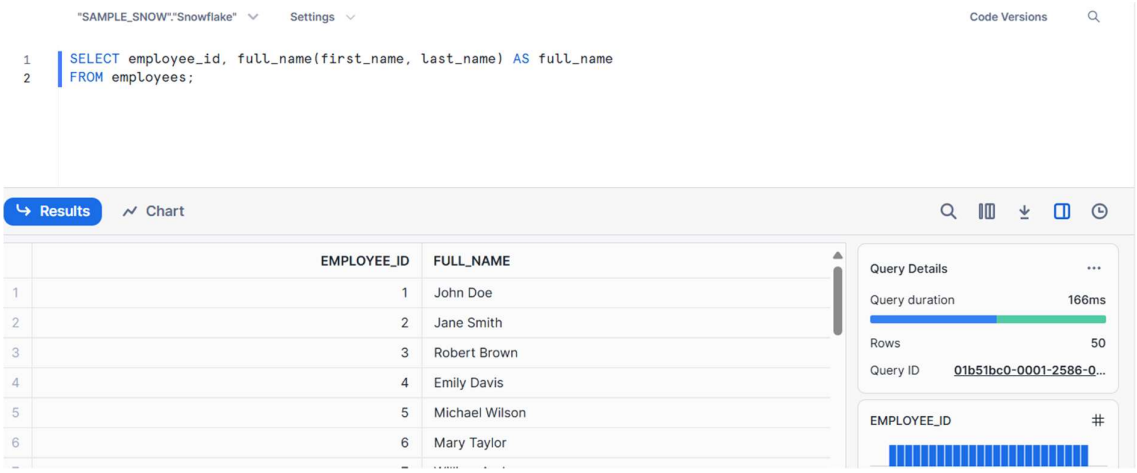
3. This UDF concatenates an employee's first name and last name with a space in between.

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



4. Concatenate first name and last name.

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



Stored Procedure

1. 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');

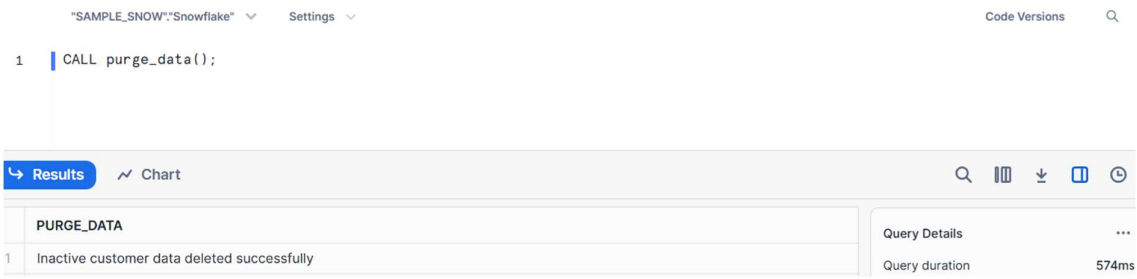
2. Check the query.

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

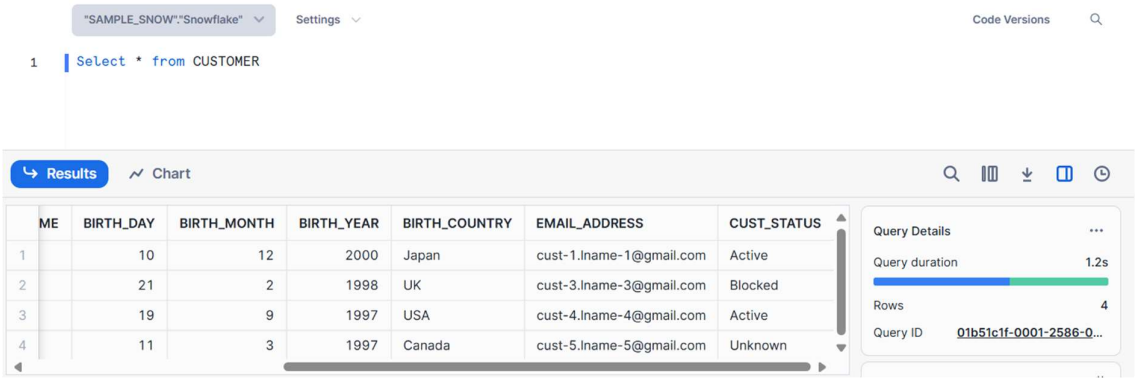
Query: CREATE OR REPLACE PROCEDURE purge_data()
RETURNS VARCHAR
LANGUAGE SQL
AS
$$
DECLARE
message VARCHAR;
BEGIN
DELETE FROM customer WHERE cust_status = 'Inactive';
message := 'Inactive customer data deleted successfully';
RETURN message;
END;
$$;

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();



6. Check the data.



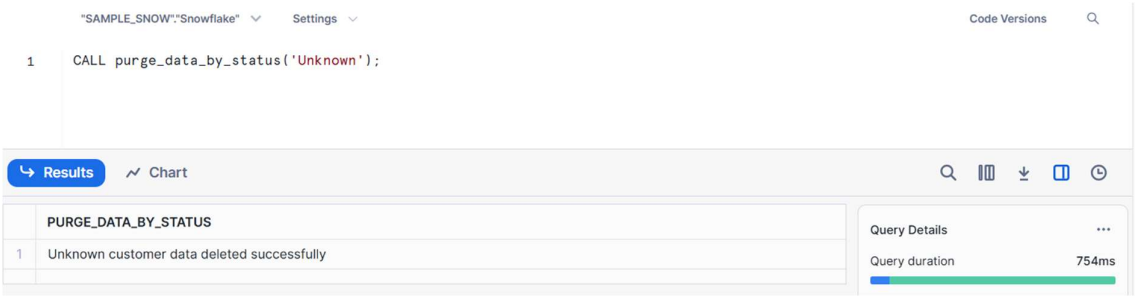
7. Create a Stored Procedure that deletes rows from the customer table where cust_status is 'Unknown'.

Query:
```
CREATE OR REPLACE PROCEDURE purge_data_by_status(in_status VARCHAR)
  RETURNS VARCHAR
  LANGUAGE SQL
AS
$$
DECLARE
  message VARCHAR;
BEGIN
  DELETE FROM customer WHERE cust_status = :in_status;
  message := in_status || ' customer data deleted successfully';
  RETURN message;
END;
$$;
```

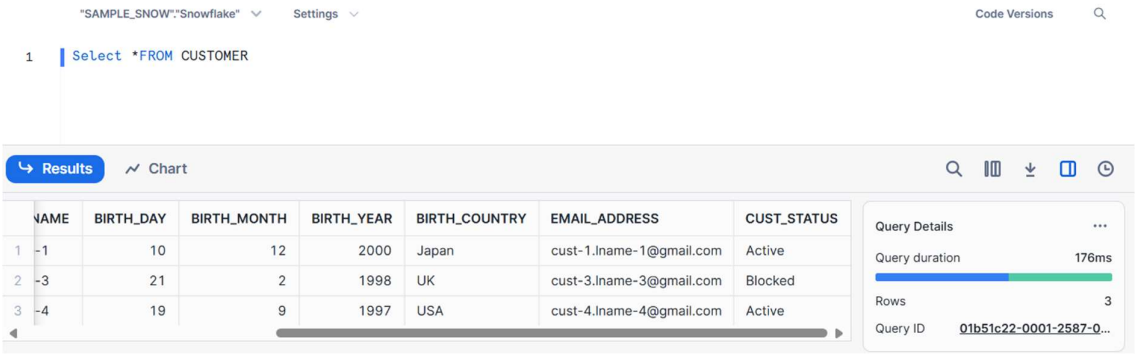
8. Check the data.

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');



6. Check the data.

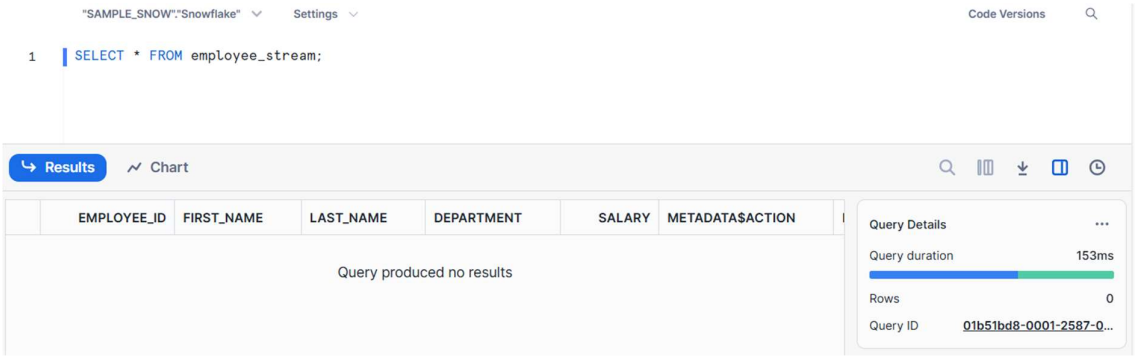


Stream

1. Creating a stream.

Query: CREATE OR REPLACE STREAM employee_stream ON TABLE employees;

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



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;

3. Enabling the Task.

Query: ALTER TASK daily_employee_summary RESUME;

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);

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.

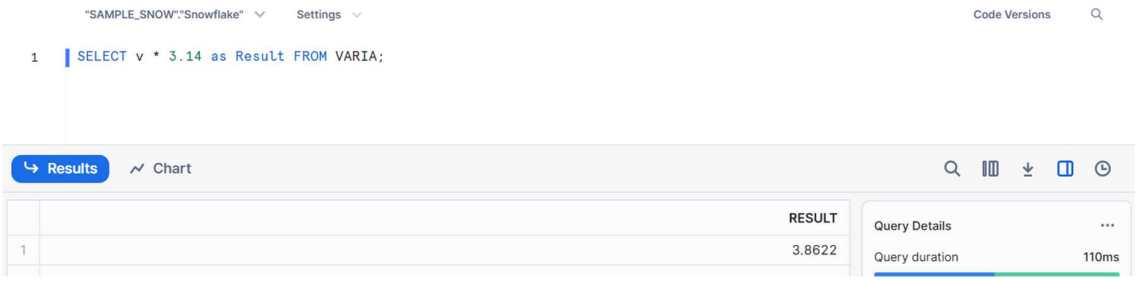
3. Check the Data.

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;

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;



Array

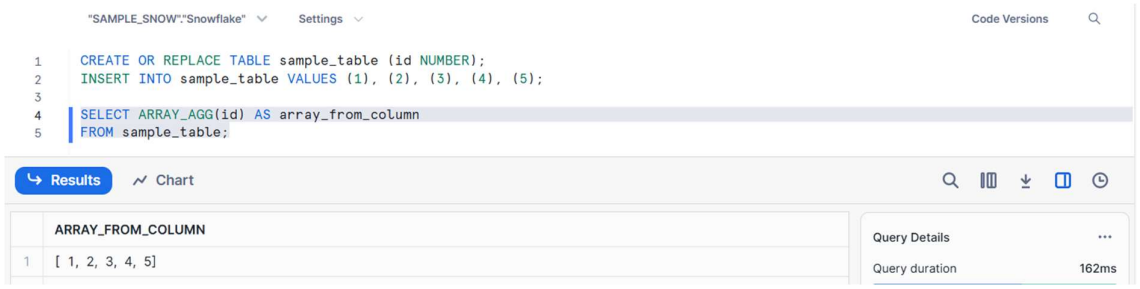
1. Create a Simple Array.

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

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;



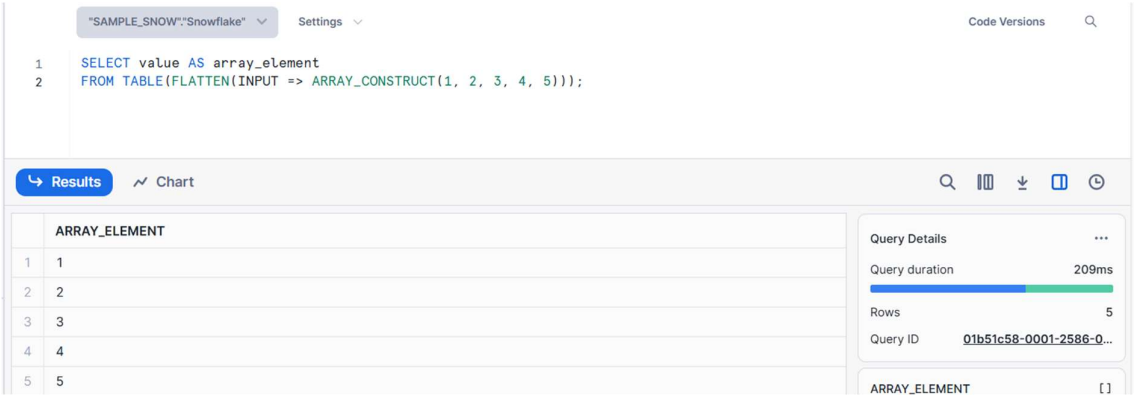
3. Accessing a Specific Element.

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

4. Accessing All Elements Using FLATTEN.

Query: SELECT value AS array_element

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



5. Appending an Element to an Array.

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

6. Prepending an Element to an Array.

Query: SELECT ARRAY_PREPEND(ARRAY_CONSTRUCT(0), ARRAY_CONSTRUCT(1, 2, 3)) AS updated_array;

Object

1. Creating a Simple Object.

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

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;

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);

4. Accessing All Key-Value Pairs Using FLATTEN.

Query: SELECT key, value

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

5. Adding a New Key-Value Pair.

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

6. Removing a Key-Value Pair.

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

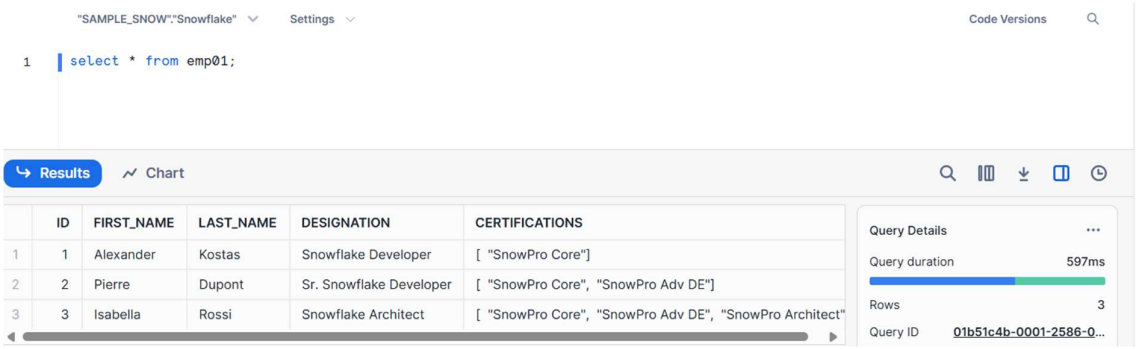
Flatten Array

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');

2. Check the data.



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

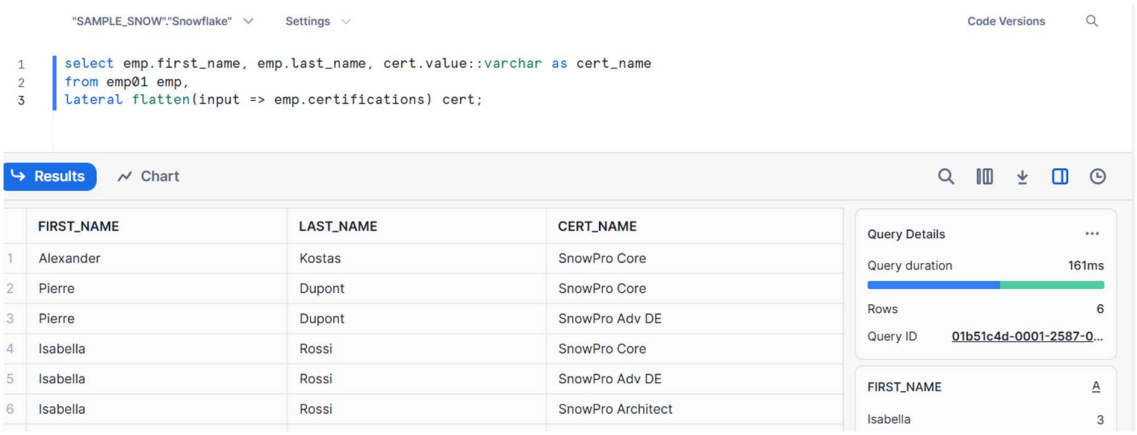
order by 1;

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;



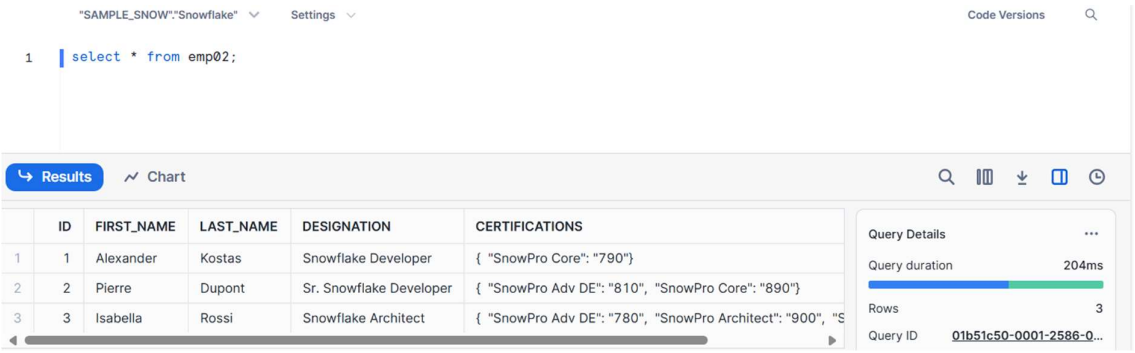
Flatten Object

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');

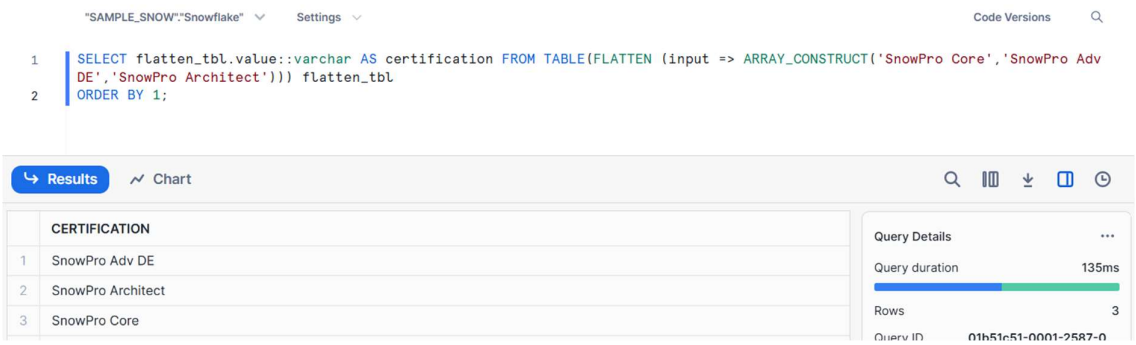
2. Check the data.



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.

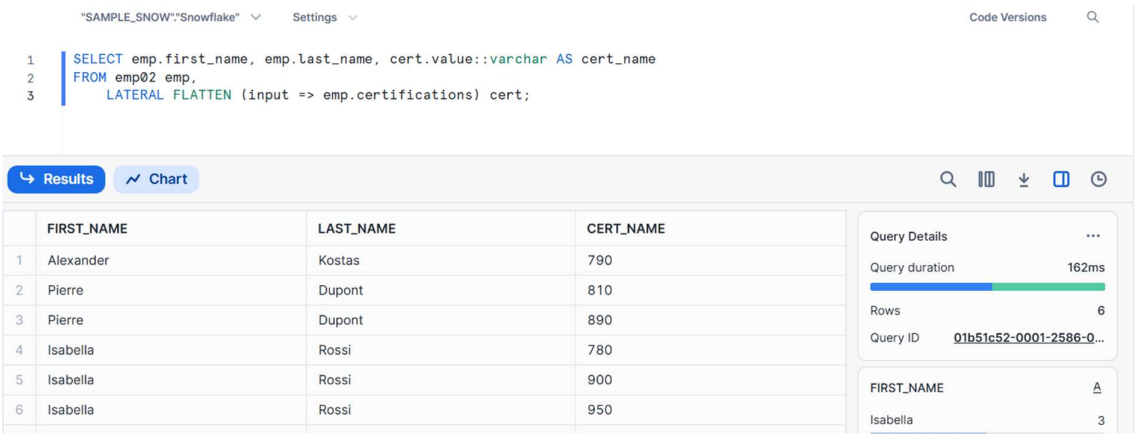
Query: SELECT flatten_tbl.value::varchar AS certification FROM TABLE(FLATTEN (input => ARRAY_CONSTRUCT('SnowPro Core','SnowPro Adv DE','SnowPro Architect'))) flatten_tbl

ORDER BY 1;



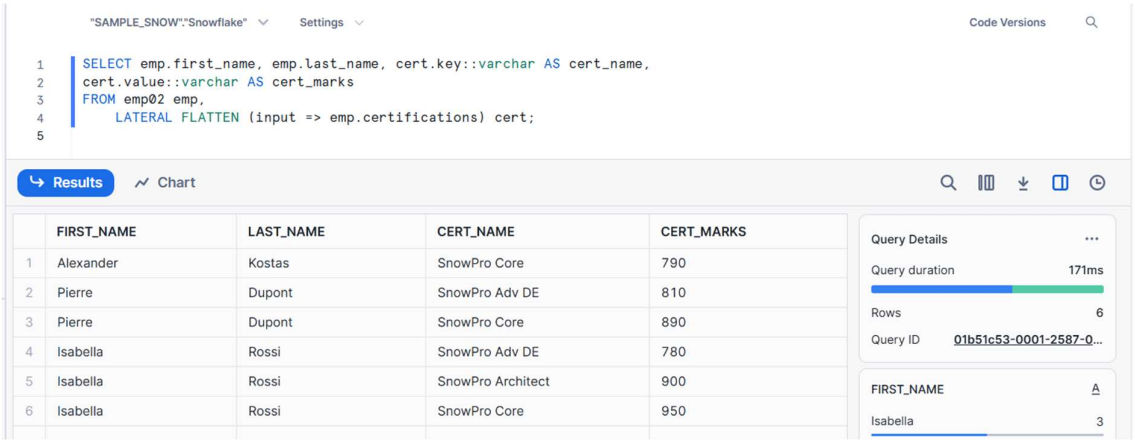
5. This below query joins each employee's record with the flattened certifications object.
6. The LATERAL FLATTEN function is used to unnest the JSON object, treating each key-value pair as a separate row.
7. The output shows each employee's first name, last name, and the names of their certifications.

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



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;



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"]}
  ]
}'),

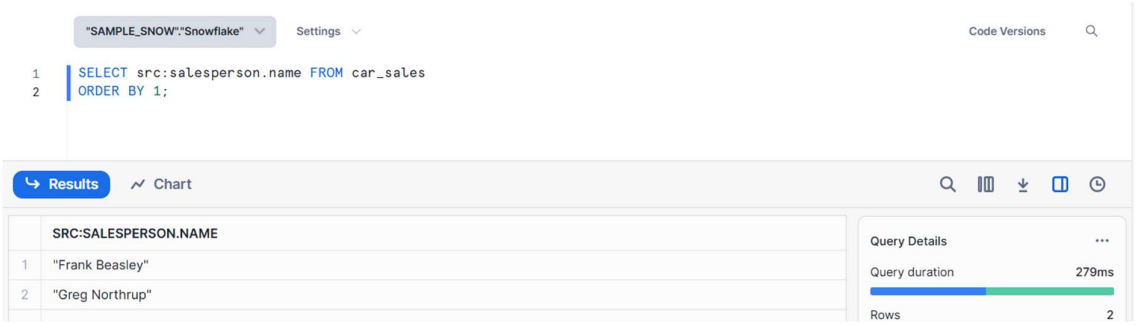
```
('{
   "date" : "2017-04-28",
   "dealership" : "Tindel Toyota",
   "salesperson" : {
    "id": "274",
    "name": "Greg Northrup"
   },
   "customer" : [
    {"name": "Bradley Greenbloom", "phone": "12127593751", "address": "New York, NY"}
   ],
   "vehicle" : [
    {"make": "Toyota", "model": "Camry",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

2. Check the Data.

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

Query: SELECT src:salesperson.name FROM car_sales

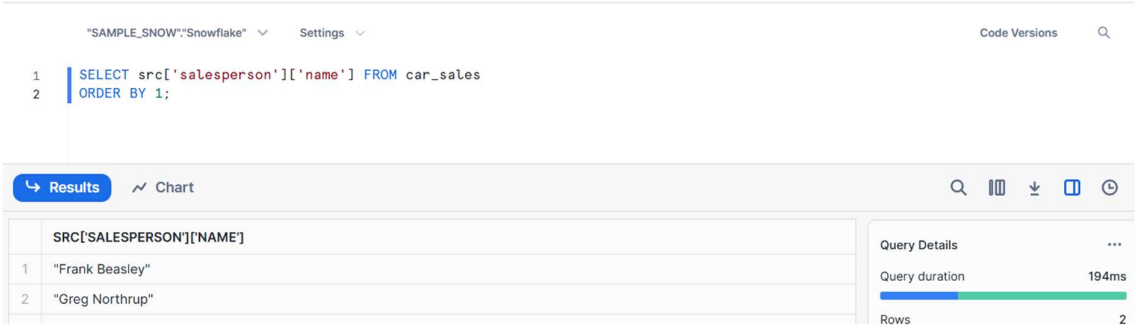
ORDER BY 1;



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

Query: SELECT src['salesperson']['name'] FROM car_sales

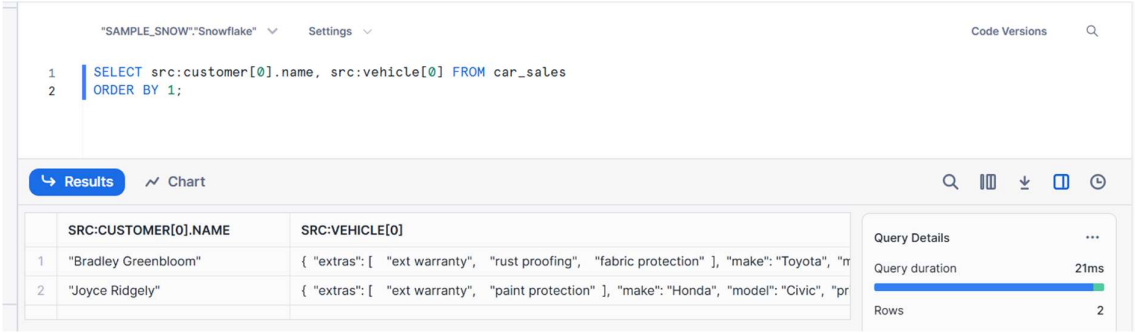
ORDER BY 1;



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;



7. Get the price of each car sold.

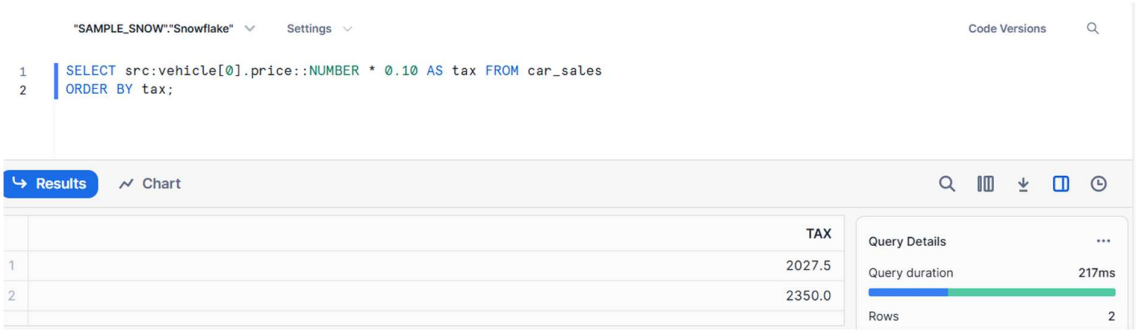
Query: SELECT src:customer[0].name, src:vehicle[0].price FROM car_sales

ORDER BY 1;

8. When you extract values from a VARIANT, you can explicitly cast the values to the desired data type.
9. 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

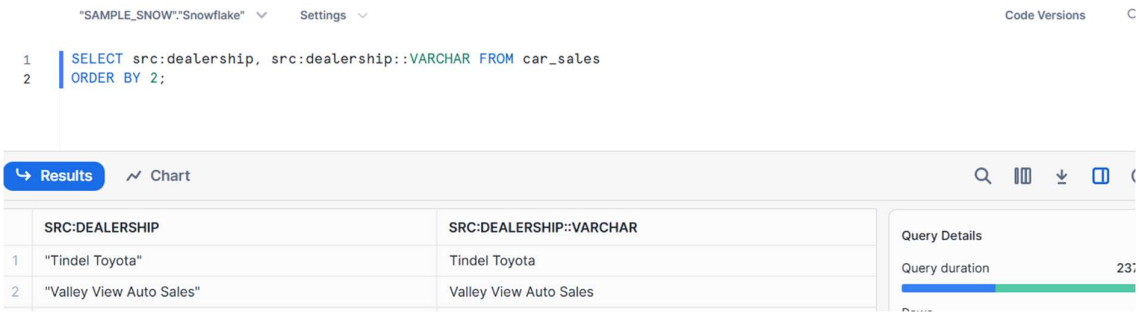
ORDER BY tax;



10. By default, when VARCHARs, DATEs, TIMEs, and TIMESTAMPs are retrieved from a VARIANT column, the values are surrounded by double quotes.
11. You can eliminate the double quotes by explicitly casting the values.

Query: SELECT src:dealership, src:dealership::VARCHAR FROM car_sales

ORDER BY 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.

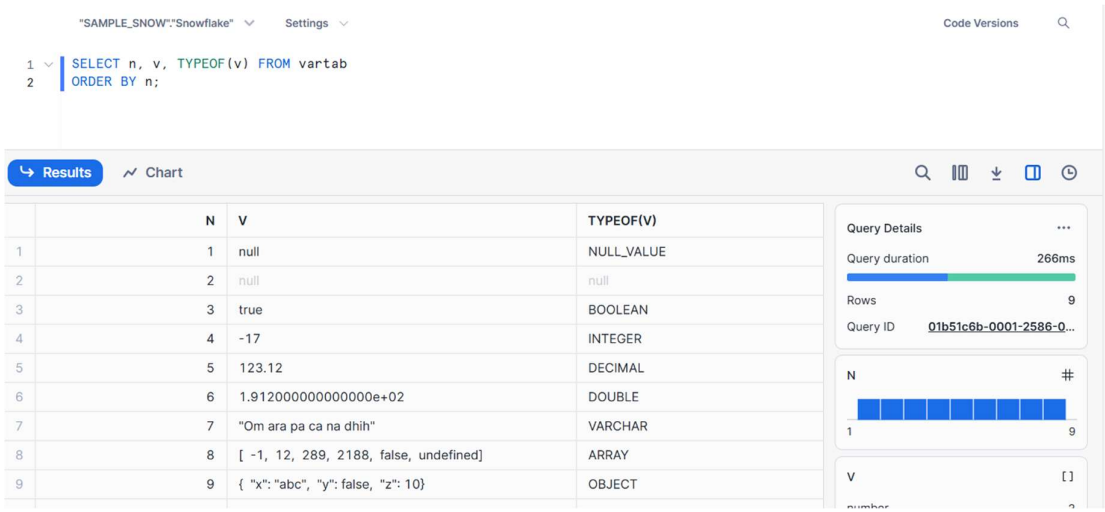
Query: CREATE OR REPLACE TABLE vartab (n NUMBER(2), v VARIANT);

```
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;
```

2. Query the Data.

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

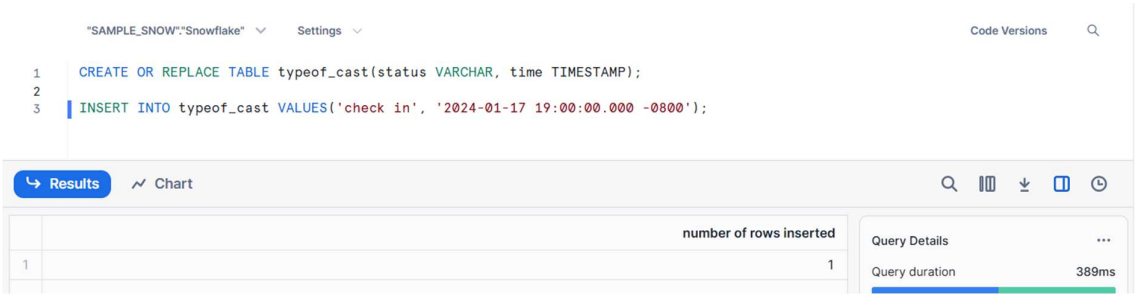
ORDER BY n;



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.

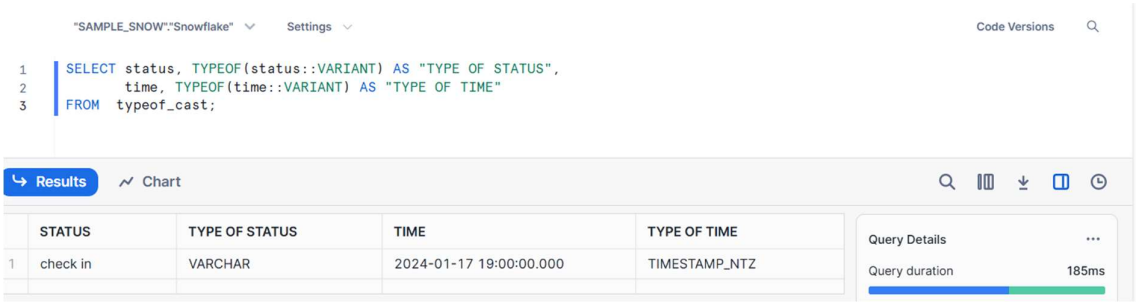
Query: CREATE OR REPLACE TABLE typeof_cast(status VARCHAR, time TIMESTAMP);

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



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;



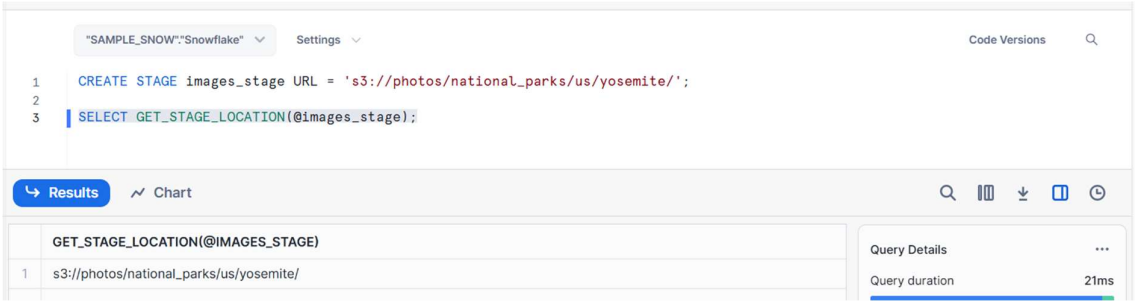
File Functions

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/';

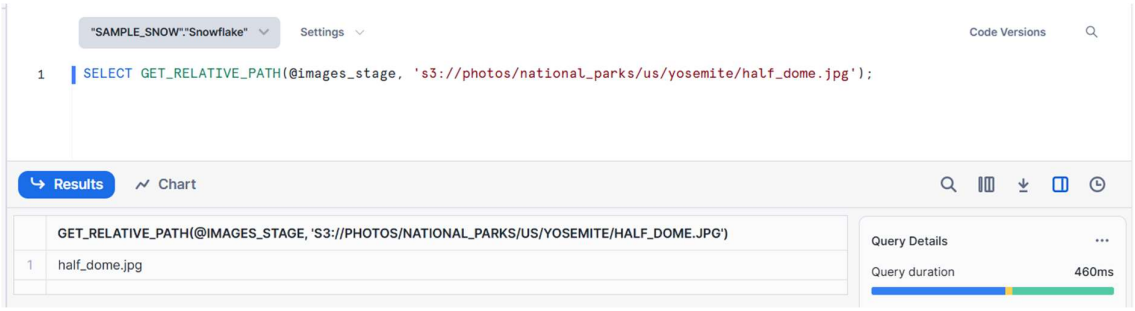
SELECT GET_STAGE_LOCATION(@images_stage);



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');



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');

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');

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');