

# Manipulating text data

DATA TYPES AND FUNCTIONS IN SNOWFLAKE



**Jake Roach**  
Field Data Engineer

# Finding the LENGTH of a string

```
LENGTH(<field>)
```

```
SELECT
```

```
    song_name,  
    LENGTH(song_name) AS characters
```

```
FROM MUSIC.songs;
```

`LENGTH` returns the numbers of characters in a string of text

- Includes spaces
- `VARCHAR` , `TEXT` , `STRING` , etc
- `LEN` is synonymous with `LENGTH`

song_name	characters
-----	-----
Levon	5
Tiny Dancer	11
Rocket Man	10

# TRIM

Removes leading and trailing characters from text values

- Most commonly used to remove spaces
- Case-insensitive
- Has counterparts `LTRIM` and `RTRIM`

```
SELECT
```

```
    <field>,
```

```
    -- Remove characters at the
```

```
    -- beginning or end of the column
```

```
    TRIM(<1>, <2>)
```

```
FROM ...;
```

`<1>` : the column or value that will be trimmed

`<2>` : *optional*, the pattern to trim from the

# TRIM

**SELECT**

song\_long\_name,

**TRIM**(song\_long\_name, '(Remastered)') **AS** trimmed\_song\_name

**FROM** MUSIC.songs;

**TRIM**(song\_long\_name, '(Remastered)')

song_long_name	trimmed_song_name
-----	-----
(Remastered) Piano Man	Piano Man
Ticking (Remastered)	Ticking
Come Sail Away	Come Sail Away

# SPLIT

Chunks text into an **array of values** based on some separator

- Return type is **ARRAY**
- Can use bracket-notation to retrieve values

**<1>** : the column to **SPLIT**

**<2>** : the separator to split by

**SELECT**

**<field>**,

-- SPLIT the field, use bracket-  
-- notation to retrieve first chunk

SPLIT(**<1>**, **<2>**),  
SPLIT(**<1>**, **<2>**)[X]

**FROM** ...;

Math, Science, Art, Reading

...

['Math', 'Science', 'Art', 'Reading']

# SPLIT

**SELECT**

collaborators,

SPLIT(collaborators, ',') **AS** all\_collaborators,

SPLIT(collaborators, ',')[0] **AS** primary\_artist *-- Return the first collaborator*

**FROM** MUSIC.songs;

collaborators	all_collaborators	primary_artist
-----	-----	-----
Queen, David Bowie	['Queen', ' David Bowie']	Queen
Elton John, Kiki Dee	['Elton John', ' Kiki Dee']	Elton John
Carly Simon, James Taylor	['Carly Simon', ' James Taylor']	Carly Simon

# CONCAT

## SELECT

```
<field>,  
<another-field>,  
<third-field>,  
  
CONCAT(  
    <field>,  
    <another-field>,  
    ' ',  
    <third-field>  
)
```

FROM ...;

Can **join** two or more text values together

- Arbitrary number of values, separated by  
,
- Need to specify spaces between characters
- Useful for combining first and last names

# CONCAT

**SELECT**

song\_name, artist\_name,

-- Concatenate three text values together

CONCAT(song\_name, ' is written by ', artist\_name) **AS** description

**FROM** MUSIC.songs;

song_name	artist_name	description
Night Moves	Bob Seger	Night Moves is written by Bob Seger
Cracklin' Rosie	Neal Diamond	Cracklin' Rosie is written by Neal Diamond



# Let's practice!

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

# Numeric calculations

DATA TYPES AND FUNCTIONS IN SNOWFLAKE



**Jake Roach**  
Field Data Engineer

# Numeric calculations

Comparison operators

Arithmetic operators

Aggregation functions and rounding

# Comparing numeric values

Comparison operators help us compare or evaluate multiple values

- `=` , are two values **equal**?
- `!=` , are two values **not equal**?
- `<` , is one value **less than** another?
- `>` , is one value **greater than** another?
- `<=` less than **or equal to**
- `>=` , greater than **or equal to**

Return `true` or `false` !

```
1 = 1,      -- true
1 != 1,     -- false
1 < 2,      -- true
1 > 2,      -- false
1 <= 2,     -- true
2 >= 2      -- true
...

```

```
WHERE 1 = 1  -- Filter records
```

# Arithmetic in Snowflake

## SELECT

```
<#> + <#>,          -- 2 + 2 -> 4
<field> - <#>,       -- 4 - 1 -> 3
<field> * <#>,       -- 3 * 2 -> 6
<field> / <#>,       -- 9 / 3 -> 3
...

```

- Can be performed between a permutation of `#` and `field`

Arithmetic operators allow us to perform "math" with numeric values

- `+` , addition
- `-` , subtraction
- `*` , multiplication
- `/` , division

# Arithmetic

**SELECT**

student\_name, exam\_score,

exam\_score + 10 AS add\_points,

exam\_score \* curve AS curved,

exam\_score / 2 AS weighted

-- Add 10 points to each student's grade

-- Curve the grade by 10%

-- Reduce the weight of the test

**FROM** STUDENTS.grades;

student_name	exam_score	add_points	curved	weighted
-----	-----	-----	-----	-----
Ryan	78	88	85.8	39
Tatiana	89	99	97.9	44.5
Pankaj	74	84	81.4	37

# Aggregation functions generate summary data

Original Table


Result Set




# Aggregation functions

```
SELECT
```

```
  <1> ,
```

```
  SUM(<field>),  -- Returns the total of a column
```

```
  AVG(<field>)   -- Finds the average value of a column
```

```
FROM ...
```

```
GROUP BY <1>;
```

Must **GROUP BY** non-aggregated fields!

- **GROUP BY ALL**



# Aggregation functions

```
SELECT
```

```
    exam_name,
```

```
    SUM(correct_answers) AS total_correct_answers,      -- Total # correct
    AVG(exam_score) AS avg_exam_score,                  -- Average exam score

    ROUND(AVG(exam_score), 1) AS rounded_exam_score    -- ROUND(<value>, <n>)
```

```
FROM STUDENTS.grades
```

```
GROUP BY exam_name;  -- GROUP BY to aggregate records, otherwise error
```

`ROUND()` takes a value to round, and the number of digits to keep after the decimal point

# Aggregation functions

exam_name	total_correct_answers	avg_exam_score	rounded_avg_exam_score
-----	-----	-----	-----
Calculus I	871	89.11111	89.1
Biology	776	87.47777	87.5
English III	541	91.33333	91.3
Python	1179	92.78787	92.8
Finance	349	96.14156	96.1

Values were generated using:

- `SUM(correct_answers)`
- `AVG(exam_score)`
- `ROUND(AVG(exam_score), 2)`

# Let's practice!

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

# Manipulating datetime data

DATA TYPES AND FUNCTIONS IN SNOWFLAKE



**Jake Roach**  
Field Data Engineer

# Extracting timestamp components

Snowflake provides functions to extract components from a timestamp

```
SELECT
    <fields>,

    DAY(<1>)  -- Many other options!

FROM ...;
```

<1> : DATE , TIME , or TIMESTAMP

- DAY
- MONTH
- YEAR
- HOUR
- MINUTE
- SECOND
- DAYNAME
- MONTHNAME

# Parsing timestamps

```
SELECT
```

```
    exam_completed_time,
```

```
    YEAR(exam_completed_time) AS y,
```

```
    MONTHNAME(exam_completed_time) AS mn,
```

```
    DAYNAME(exam_completed_time) AS dn,
```

```
    DAY(exam_completed_time) AS d,
```

```
    HOUR(exam_completed_time) AS h,
```

```
    MINUTE(exam_completed_time) AS m,
```

```
    SECOND(exam_completed_time) AS s
```

```
FROM STUDENTS.grades;
```

# Parsing timestamps

exam_completed_time	y	mn	dn	d	h	m	s
-----	-----	-----	-----	-----	-----	-----	-----
2025-06-25 11:17:02	2025	Jun	Wed	25	11	6	2
2025-06-26 08:49:49	2025	Jun	Thu	26	8	49	49
2025-06-25 09:56:07	2025	Jun	Wed	25	9	56	7
2025-06-24 22:14:27	2025	Jun	Tue	24	22	14	27
2025-06-26 13:36:55	2025	Jun	Thu	26	13	36	55
2025-06-25 16:23:09	2025	Jun	Wed	25	16	23	9
...							

# DATEDIFF

**DATEDIFF** finds the interval between two dates or timestamps

**SELECT**

<fields> ,

DATEDIFF(<1>, <2>, <3>) -- unit, first timestamp, second timestamp

**FROM** ...;

<1> : Unit of time the result will be in, such as **MINUTE** , **HOUR** , **DAY** , **YEAR** , **WEEK** , etc.

<2> : Starting timestamp

<3> : Ending timestamp



# Using DATEDIFF

First TIMESTAMP: 2025-05-12 08:24:08

Second TIMESTAMP: 2025-11-13 03:05:46

```
SELECT
  DATEDIFF(
    DAY,
    TO_DATE('2025-05-12 08:24:08'),
    TO_DATE('2025-11-13 03:05:46')
  )
```

185 (DAYS)

# DATEADD

SELECT

```
-- unit, number of units, timestamp
DATEADD(
  DAY,
  185,
  TO_DATE('2025-05-12 08:24:08')
)
```

FROM ...;

2025-05-12 08:24:08 + 185 DAYS

= 2025-11-13 08:24:08

Add intervals of time to `DATE` , `TIME` , or `TIMESTAMP` using `DATEADD`

<1> : Unit of time to add, such as `MINUTE` , `HOUR` , `DAY` , `YEAR` , `WEEK` , etc.

<2> : # of "units" to add to timestamp

<3> : The timestamp that will be added to

# Manipulate a DATE

```
SELECT
```

```
  s_id,
```

```
  exam_completed_time AS completed_time,
```

```
  exam_due_time AS due_time,
```

```
  -- Find the difference in time between completion and due date
```

```
  DATEDIFF(HOUR, exam_completed_time, exam_due_time) AS hours_early,
```

```
  -- Determine date that teacher must complete grading by
```

```
  DATEADD(WEEK, 1, exam_completed_time) AS grading_due
```

```
FROM STUDENTS.grades;
```

# Manipulate a DATE

s_id	completed_time	due_time	hours_early	grading_due
-----	-----	-----	-----	-----
919	2025-06-25 11:17:02	2025-06-26 10:00:00	22	2025-07-02 11:17:02
871	2025-06-26 08:49:49	2025-06-26 10:00:00	1	2025-07-03 08:49:49
111	2025-06-25 09:56:07	2025-06-26 10:00:00	24	2025-07-02 09:56:07
465	2025-06-24 22:14:27	2025-06-26 10:00:00	35	2025-07-01 22:14:27
248	2025-06-26 13:36:55	2025-06-26 10:00:00	-3	2025-07-23 13:36:55
767	2025-06-25 16:23:09	2025-06-26 10:00:00	17	2025-07-02 16:23:09
...				

# Let's practice!

DATA TYPES AND FUNCTIONS IN SNOWFLAKE

# User-defined functions in Snowflake

DATA TYPES AND FUNCTIONS IN SNOWFLAKE



**Jake Roach**  
Field Data Engineer

# What are user-defined functions (UDFs)?

"Package" complex logic into a single,  
**reusable function**

- Easier to read
- Reduces chances for mistakes
- Share logic with others

... let's take a closer look!

SELECT

```
CONCAT(  
  first_name,  
  '.',  
  last_name,  
  ...  
)
```

```
build_email_address
```



FROM ... ;

# Refactoring code with a UDF

**SELECT**

first\_name, last\_name, school\_name,

CONCAT(

first\_name,

'.',

last\_name,

'@',

school\_name,

'.com'

)

**FROM** STUDENTS.personal\_info;

**SELECT**

first\_name,

last\_name,

school\_name,

-- Reusable, easy to read

build\_email\_address(

first\_name,

last\_name,

school\_name

) **AS** email\_address

**FROM** STUDENTS.personal\_info;



# Defining a UDF

```
CREATE OR REPLACE FUNCTION <name> (  
  -- Specify arguments and types  
  <arg1> <type1>,  
  <arg2> <type2>,  
  ...  
)  
  
RETURN <type>  -- Declare return type  
  
AS  
  
$$  
-- Define your function here  
$$
```

1. CREATE OR REPLACE FUNCTION
2. Provide a function name
3. In parentheses, specify the **name** and **type** of each argument
4. Declare the RETURN type
5. Define your logic in \$\$ ... \$\$ !

# Building an email address

```
CREATE OR REPLACE FUNCTION build_email_address(  
  first_name TEXT,  
  last_name TEXT,  
  school_name TEXT  
)  
  
RETURN TEXT  
  
AS  
  
$$  
CONCAT(LOWER(first_name), '.', LOWER(last_name), '@', LOWER(school_name), '.com')  
$$;
```

# Using a UDF

**SELECT**

```
first_name, last_name, school_name,  
build_email_address(first_name, last_name, school_name) AS email_address
```

**FROM** STUDENTS.personal\_info;

first_name	last_name	school_name	email_address
-----	-----	-----	-----
Ryan	Cohen	Harvard	ryan.cohen@harvard.com
Tatiana	Doyle	Stanford	tatiana.doyle@stanford.com
Pankaj	Pandey	MIT	pankaj.pandey@mit.com
Jake	Roach	Purdue	jake.roach@purdue.com

# Let's practice!

DATA TYPES AND FUNCTIONS IN SNOWFLAKE