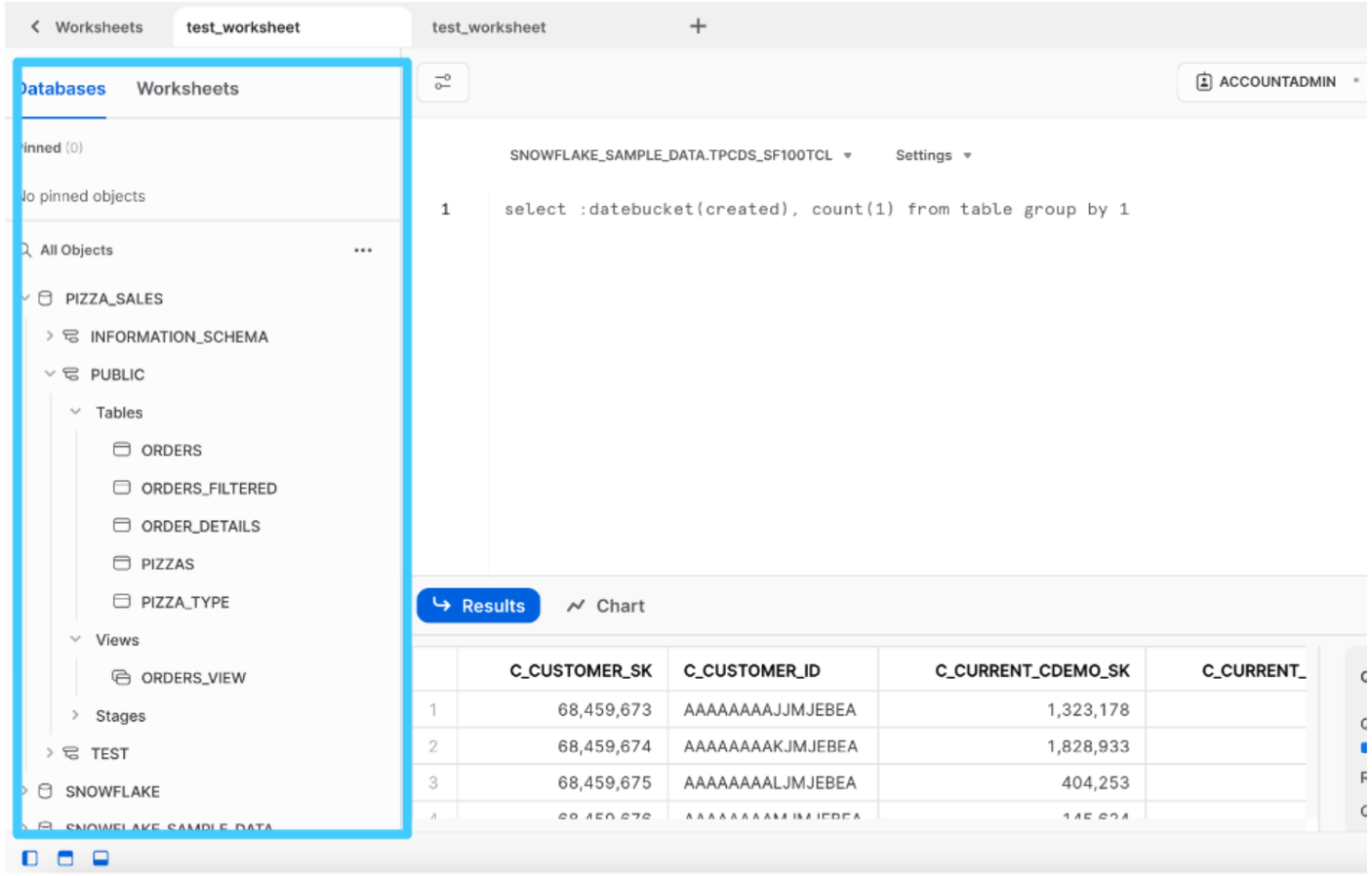
Connecting to Snowflake and DDL commands

INTRODUCTION TO SNOWFLAKE

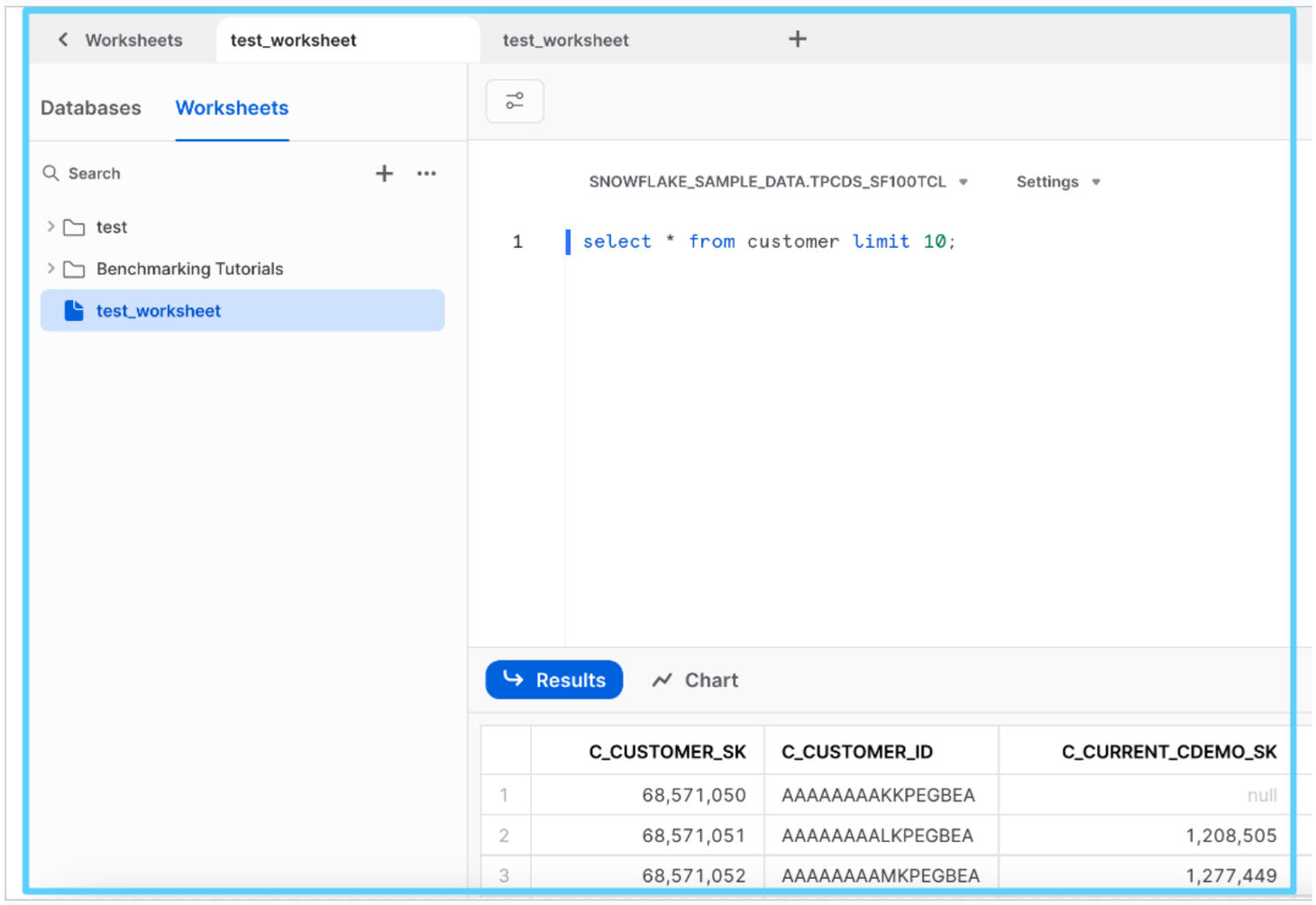
Palak Raina
Senior Data Engineer

Connecting to Snowflake

- Snowsight: Snowflake Web Interface



Web Interface: Worksheets



Connecting to Snowflake: Drivers, SnowSQL

Drivers & Connectors

- ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity) drivers
- Connectors : Python/Spark and more

SnowSQL

- Command-line client
  - Installed on Linux, Windows, or Mac

Staging

- Temporary location storing data
- Internal Stage
- External Stage (Cloud storage: Amazon S3, Google Cloud Storage)
  - **Raw data sources:** Initial, unprocessed data, e.g., CSV files
  - **Staging Area:** Temporary storage, loading
  - **Table:** Final data is loaded here

Staging

CREATE STAGE

```
CREATE STAGE my_local_stage
```

```
PUT file:///path_to_your_local_file/orders.csv
@my_local_stage  -- stage name prefixed with @
```

@ -> Prefix to reference stage.

DDL Commands

- Some Data Definition Language (DDL) commands
  - CREATE
  - ALTER
  - DROP
  - RENAME
  - COMMENT

DDL: Create

```
CREATE TABLE orders_pizza (
  order_id NUMBER,
  order_date DATE,
  time TIME
)
```

```
CREATE VIEW orders_pizza_view AS
SELECT order_id,
       order_date
FROM orders_pizza
```

DDL: ALTER, RENAME, DROP

- ALTER & RENAME

```
ALTER TABLE IF EXISTS orders_pizza RENAME TO orders;
```

- `IF_EXISTS` -> Table's presence

```
ALTER TABLE orders
RENAME COLUMN time TO order_time;
```

- DROP

```
DROP TABLE orders
```

DDL: Comment

```
CREATE TABLE pizza_type (
  pizza_type_id VARCHAR(50) COMMENT 'Unique identifier for pizza type',
  name VARCHAR(100) ,
  category VARCHAR(50),
  ingredients VARCHAR(500)
  )
COMMENT = 'Table that stores information about different types of pizzas, including their names, categories, and ingredients'
```

Postgres

```
COMMENT ON [OBJECT TYPE] [OBJECT NAME] IS 'comment';
COMMENT ON TABLE pizza_type IS 'Table with pizza type info'
```

Let's practice!

INTRODUCTION TO SNOWFLAKE

Snowflake database structures and DML

INTRODUCTION TO SNOWFLAKE

Palak Raina
Senior Data Engineer

datacamp

Overview

Snowflake

- `SHOW`
- `DESCRIBE`
- `INSERT`
- `UPDATE`
- `MERGE`
- `COPY`

Similarities to Postgres

- `INSERT`
- `UPDATE`
- `MERGE`

SHOW DATABASES

Snowflake

- `SHOW`

```
SHOW DATABASES
```

created_on	name	is_default	is_current	origin
2023-08-06 11:45:34.203 -0700	PIZZA_SALES	N	Y	
2023-08-06 07:10:36.999 -0700	SNOWFLAKE	N	N	SNOWFLAKE.ACCOUNT_USAGE
2023-08-06 07:10:39.622 -0700	SNOWFLAKE_SAMPLE_	N	N	SFSALESSHARED.SFC_SAMPLES_GCP_EUROPE_WEST4.SAMPLE_DATA
2023-08-06 07:43:21.450 -0700	TEST_DB	N	N	
2023-08-06 11:55:51.279 -0700	UBER_DATA	N	N	
2023-08-06 12:06:47.020 -0700	YELP_DATA	N	N	

SHOW TABLES

```
SHOW TABLES IN { DATABASE [ <db_name> ] }
```

```
SHOW TABLES IN DATABASE PIZZA_SALES
```

created_on	name	database_name	schema_name	comment
2023-08-07 12:24:13.041 -0700	ORDERS	PIZZA_SALES	PUBLIC	
2023-08-09 07:16:16.471 -0700	ORDERS_FILTERED	PIZZA_SALES	PUBLIC	
2023-08-06 11:49:00.436 -0700	ORDER_DETAILS	PIZZA_SALES	PUBLIC	
2023-08-06 11:47:55.545 -0700	PIZZAS	PIZZA_SALES	PUBLIC	
2023-08-07 14:33:01.431 -0700	PIZZA_TYPE	PIZZA_SALES	PUBLIC	Table that stores information about different types of pizzas, including their names, categories, an

SHOW TABLES LIKE

```
SHOW TABLES [ LIKE '<pattern>' ]
            [ IN { DATABASE [ <db_name> ] } ]
```

```
SHOW TABLES LIKE '%PIZZA%' IN DATABASE PIZZA_SALES
```

created_on	name	database_name	schema_name	comment
2023-08-06 11:47:55.545 -0700	PIZZAS	PIZZA_SALES	PUBLIC	
2023-08-07 14:33:01.431 -0700	PIZZA_TYPE	PIZZA_SALES	PUBLIC	Table that stores information about different types of pizzas,

SHOW SCHEMAS, COLUMNS

```
SHOW SCHEMAS IN DATABASE PIZZA_SALES
```

created_on	name	is_default	is_current	database_name	owner	comment	options
2023-08-09 04:24:17.700 -0700	INFORMATION_SCHEMA	N	N	PIZZA_SALES		Views describing the contents of schemas in this database	
2023-08-06 11:45:34.213 -0700	PUBLIC	N	Y	PIZZA_SALES	ACCOUNTADMIN		
2023-08-09 03:53:10.391 -0700	TEST	N	N	PIZZA_SALES	ACCOUNTADMIN		

```
SHOW COLUMNS IN PIZZA_TYPE
```

table_name	schema_name	column_name	data_type	null?	default	kind	expression	comment
PIZZA_TYPE	PUBLIC	PIZZA_TYPE_ID	{"type":"TEXT","length":50,"byteLength":200,"nullable":false,"fixed":false}	NOT_NULL		COLUMN		Unique identifier for pizza type
PIZZA_TYPE	PUBLIC	NAME	{"type":"TEXT","length":100,"byteLength":400,"nullable":true,"fixed":false}	true		COLUMN		
PIZZA_TYPE	PUBLIC	CATEGORY	{"type":"TEXT","length":50,"byteLength":200,"nullable":true,"fixed":false}	true		COLUMN		Categorization of the pizza based on style, origin, or other criteria
PIZZA_TYPE	PUBLIC	INGREDIENTS	{"type":"TEXT","length":500,"byteLength":2000,"nullable":true,"fixed":false}	true		COLUMN		Comma-separated list of ingredients included in the pizza

SHOW VIEWS

```
SHOW VIEWS IN DATABASE PIZZA_SALES
```

··· created_on	name	reserved	database_name	schema_name	owner	comment
1969-12-31 16:00:00.000 -0800	APPLICABLE_ROLES		PIZZA_SALES	INFORMATION_SCHEMA		The roles that can be applied to the current user.
1969-12-31 16:00:00.000 -0800	COLUMNS		PIZZA_SALES	INFORMATION_SCHEMA		The columns of tables defined in this database that are accessible to the cu
1969-12-31 16:00:00.000 -0800	DATABASES		PIZZA_SALES	INFORMATION_SCHEMA		The databases that are accessible to the current user's role.
1969-12-31 16:00:00.000 -0800	ENABLED_ROLES		PIZZA_SALES	INFORMATION_SCHEMA		The roles that are enabled to the current user.
1969-12-31 16:00:00.000 -0800	EVENT_TABLES		PIZZA_SALES	INFORMATION_SCHEMA		The event tables defined in this database that are accessible to the current
1969-12-31 16:00:00.000 -0800	EXTERNAL_TABLES		PIZZA_SALES	INFORMATION_SCHEMA		The external tables defined in this database that are accessible to the curre
1969-12-31 16:00:00.000 -0800	FILE_FORMATS		PIZZA_SALES	INFORMATION_SCHEMA		The file formats defined in this database that are accessible to the current u
1969-12-31 16:00:00.000 -0800	FUNCTIONS		PIZZA_SALES	INFORMATION_SCHEMA		The user-defined functions defined in this database that are accessible to tl

DESCRIBE DATABASE, SCHEMA

- `DESCRIBE` or `DESC`

```
DESCRIBE DATABASE PIZZA_SALES
```

created_on	name	...	kind
2023-08-09 04:45:45.718 -0700	INFORMATION_SCHEMA		SCHEMA
2023-08-06 11:45:34.213 -0700	PUBLIC		SCHEMA
2023-08-09 03:53:10.391 -0700	TEST		SCHEMA

```
DESCRIBE SCHEMA PUBLIC
```

created_on	name	kind
2023-08-07 12:24:13.041 -0700	ORDERS	TABLE
2023-08-07 12:16:31.776 -0700	ORDERS_VIEW	VIEW
2023-08-06 11:49:00.436 -0700	ORDER_DETAILS	TABLE
2023-08-06 11:47:55.545 -0700	PIZZAS	TABLE

DESCRIBE TABLE, VIEW

```
DESCRIBE TABLE PIZZA_TYPE
```

name	type	kind	null?	primary key	comment
PIZZA_TYPE_ID	VARCHAR(50)	COLUMN	N	Y	Unique identifier for pizza type
NAME	VARCHAR(100)	COLUMN	Y	N	null
CATEGORY	VARCHAR(50)	COLUMN	Y	N	Categorization of the pizza based on style, origin, or other criteria
INGREDIENTS	VARCHAR(500)	COLUMN	Y	N	Comma-separated list of ingredients included in the pizza

```
DESCRIBE VIEW ORDERS_VIEW
```

name	type	kind	null?	primary key
ORDER_ID	NUMBER(38,0)	COLUMN	N	N
ORDER_DATE	DATE	COLUMN	Y	N

DESCRIBE STAGE

```
DESCRIBE STAGE my_local_stage
```

parent_property	property	property_type	property_value ···	property_default
STAGE_FILE_FORMAT	TYPE	String	CSV	CSV
STAGE_FILE_FORMAT	RECORD_DELIMITER	String	\n	\n
STAGE_FILE_FORMAT	FIELD_DELIMITER	String	,	,
STAGE_FILE_FORMAT	FILE_EXTENSION	String		
STAGE_FILE_FORMAT	SKIP_HEADER	Integer	0	0
STAGE_FILE_FORMAT	PARSE_HEADER	Boolean	false	false

DML (Data Manipulation Language) Commands

INSERT

- Insert Using Explicitly Specified Values

```
INSERT INTO orders (order_id, order_date, order_time)
VALUES (1, '2015-01-01', '11:38:36')
```

...	ORDER_ID	ORDER_DATE	ORDER_TIME
	1	2015-01-01	11:38:36

INSERT Using Query

- Insert using Query

```
INSERT INTO orders_filtered
  SELECT * FROM orders
      WHERE order_date > '2015-01-02'
```

ORDER_ID	ORDER_DATE	TIME
3	2015-01-03	14:20:20
4	2015-01-04	09:10:55
5	2015-01-05	16:38:50

UPDATE

```
UPDATE orders
SET order_time = '17:00:00'
WHERE order_id = '5'
```

Before:

ORDER_ID	ORDER_DATE	ORDER_TIME
5	2015-01-05	16:38:50

After:

ORDER_ID	ORDER_DATE	ORDER_TIME
5	2015-01-05	17:00:00

MERGE

- Combines data from two tables

```
MERGE INTO orders_filtered AS target -- Target table
USING orders AS source -- Source table
ON target.order_id = source.order_id -- Common column
WHEN MATCHED THEN -- When there is a match
 UPDATE SET
 -- Update order_date and time of target table
  target.order_date = source.order_date,
  target.time = source.order_time
```

MERGE RESULTS

Source table: `orders`

ORDER_ID	ORDER_DATE	ORDER_TIME
1	2015-01-01	11:38:36
2	2015-01-02	12:15:45
3	2015-01-03	14:20:20
4	2015-01-04	09:10:55
5	2015-01-05	17:00:00

Before Merge: `orders_filtered`

ORDER_ID	ORDER_DATE	TIME
3	2015-01-03	14:20:20
4	2015-01-04	09:10:55
5	2015-01-05	14:38:50

After Merge: `orders_filtered` updated `order_id = 5` based on `orders` table

ORDER_ID	ORDER_DATE	TIME
3	2015-01-03	14:20:20
4	2015-01-04	09:10:55
5	2015-01-05	17:00:00

COPY

Snowflake:

```
COPY INTO orders FROM  @my_local_stage/orders.csv
FILE_FORMAT = (TYPE = 'CSV' SKIP_HEADER=1 )
```

- `@my_local_stage` : stage we have created.
- `orders.csv` : file within that stage we're copying data from.
- `FILE_FORMAT` : format of the source data, in this case, a CSV.

Let's practice!

INTRODUCTION TO SNOWFLAKE

Snowflake data type and data type conversion

INTRODUCTION TO SNOWFLAKE

Palak Raina
Senior Data Engineer

datacamp

Data types

- `VARCHAR`
- `NUMERIC`
- `INT`
- `DATE`
- `TIME`
- `TIMESTAMP`
- `VARIANT` -> Semi-structured data

Similarities with Postgres known data types (non-exhaustive list):

- `VARCHAR`
- `NUMERIC`
- `INT`

Comparison with Postgres

Data Type	Snowflake	PostgreSQL
VARCHAR Max Length	16,777,216	65,535
NUMERIC Default Precision	38	37
INTEGER Range	~±10^37	32-bit signed range

DATE

- `DATE`
  - Formats: `YYYY-MM-DD` , `DD-MM-YYYY` , etc.
  - Default: `YYYY-MM-DD`

```
CREATE TABLE orders (
  order_id NUMBER ,
  order_date DATE -- DATE data type
  )
```

ORDER_ID	ORDER_DATE
1	2015-01-01
2	2015-01-02
3	2015-01-03
4	2015-01-04
5	2015-01-05

TIME

- `TIME`
  - Format: `HH:MI:SS`

```
CREATE TABLE orders (
  order_id NUMBER ,
  order_date DATE,
  order_time TIME -- TIME data type
  )
```

ORDER_ID	ORDER_DATE	ORDER_TIME
1	2015-01-01	11:38:36
2	2015-01-02	12:15:45
3	2015-01-03	14:20:20

TIMESTAMP

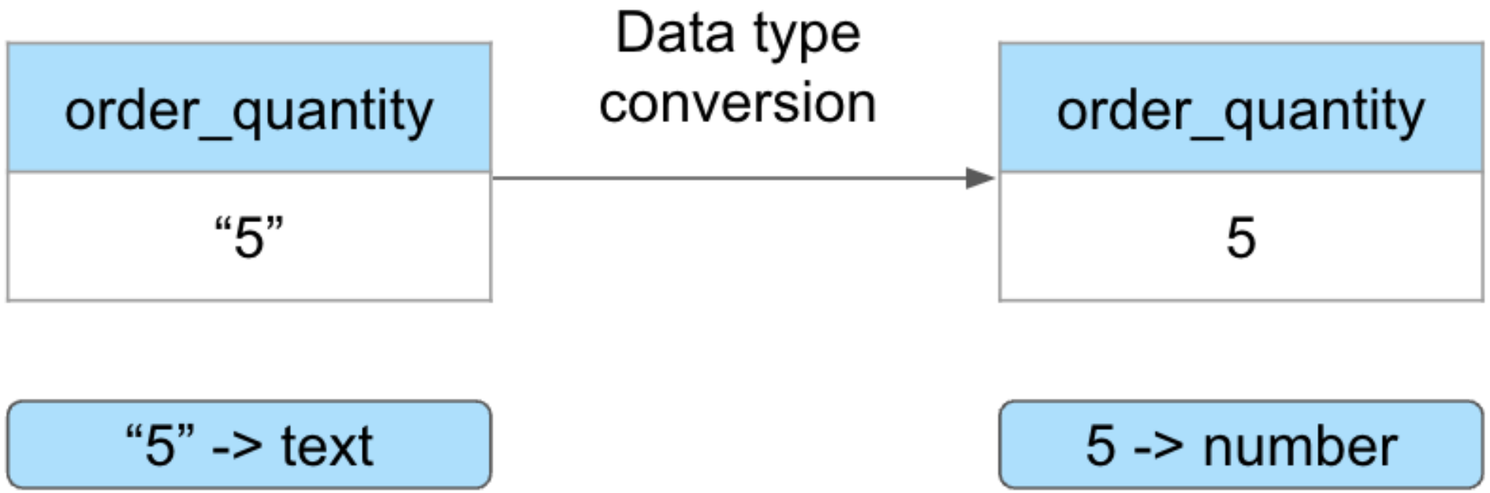
- `TIMESTAMP` - Combines `DATE` and `TIME`
  - Format: `YYYY-MM-DD HH:MI:SS`

```
CREATE TABLE orders (
  order_id NUMBER ,
  order_timestamp TIMESTAMP -- Timestamp
  )
```

ORDER_ID	ORDER_TIMESTAMP
1	2015-01-01 00:00:00
2	2015-01-02 00:00:00
3	2015-01-03 00:00:00
4	2015-01-04 00:00:00

Data type conversion - What?

- Converting data from one type to another



Data type conversion - Why?

- Improving performance
- Data accuracy and consistency
- Data quality

Data type conversion - How?

1. CAST *Syntax:*
   - `CAST( <source_data/column> AS <target_data_type> )`
   - `CAST('80' AS INT)`
2. :: *Syntax:*
   - `<source_data/column>::<target_data_type>`
   - `'80'::INT`

CAST COLUMN

```
SELECT CAST(order_date AS TIMESTAMP)
       AS order_timestamp
FROM orders
```

Before casting:

ORDER_DATE
2015-01-01
2015-01-02
2015-01-03
2015-01-04
2015-01-05

After casting:

ORDER_TIMESTAMP
2015-01-01 00:00:00
2015-01-02 00:00:00
2015-01-03 00:00:00
2015-01-04 00:00:00
2015-01-05 00:00:00

Conversion functions

- Examples: `TO_VARCHAR` , `TO_DATE` , etc.

TO_DATE

- `TO_DATE( <expr> )`
  - expr - string, timestamp, etc.
  - Result: DATE

Example:

```
SELECT TO_DATE('2023-08-16 11:51:00')
```

Result:

`2023-08-16`

Let's practice!

INTRODUCTION TO SNOWFLAKE

Functions, sorting, and grouping

INTRODUCTION TO SNOWFLAKE

Palak Raina
Senior Data Engineer

Functions

- `AGGREGATE`
- `STRING`
- `DATE` & `TIME`

Aggregate functions

Aggregation	Command	Example
Averaging	`AVG()`	`SELECT AVG( <expr> ) FROM table`
Sum	`SUM()`	`SELECT SUM( <expr> ) FROM table`
Minimum Value	`MIN()`	`SELECT MIN( <expr> ) FROM table`
Maximum Value	`MAX()`	`SELECT MAX( <expr> ) FROM table`
Count	`COUNT()`	`SELECT COUNT( <expr> ) FROM table`

String functions - CONCAT

-- Complete the CONCAT function for columns pickup_point and status

SELECT CONCAT('Trip from ', pickup_point, ' was completed with status: ', status) AS trip_comment
FROM uber_request_data

- Combines the expressions.

Syntax:

```
CONCAT( <expr1> [ , <exprN> ... ] )
```

- Combining `category` with ' - Pizza'

```
SELECT CONCAT(category, ' - Pizza')
  AS pizza_category
FROM pizza_type
```

Before Concat:

CATEGORY
Chicken
Classic
Supreme
Veggie

After Concat:

PIZZA_CATEGORY
Chicken - Pizza
Classic - Pizza
Supreme - Pizza
Veggie - Pizza

UPPER & LOWER

Syntax: `UPPER( <expr> )`

```
SELECT UPPER(category) AS upper_category
FROM pizza_type
```

UPPER_CATEGORY
CHICKEN
CLASSIC
SUPREME
VEGGIE

Syntax: `LOWER( <expr> )`

```
SELECT LOWER(category) AS lower_category
FROM pizza_type
```

LOWER_CATEGORY
chicken
classic
supreme
veggie

DATE & TIME functions

- `CURRENT_DATE()` or `CURRENT_DATE`
- `CURRENT_TIME()` or `CURRENT_TIME`

```
SELECT CURRENT_DATE
```

```
SELECT CURRENT_TIME
```

CURRENT_DATE
2023-08-15

CURRENT_TIME
13:35:58

EXTRACT

Syntax

- `EXTRACT( <date_or_time_part> FROM <date_or_time_expr> )`
  - `<date_or_time_part>` - `year`, `month`, `day`, etc.

```
SELECT drop_timestamp,
       EXTRACT(YEAR FROM drop_timestamp) AS year
FROM uber_data
```

DROP_TIMESTAMP	YEAR
2016-11-07 13:00:00.000	2,016

SORTING and GROUPING

- SORTING: `ORDER BY`
- GROUPING: `GROUP BY`
  - Snowflake: `GROUP BY ALL`

GROUP BY ALL

- `GROUP BY column1, column2`

```
SELECT
    pizza_type_id,
    size,
    AVG(price) AS average_price
FROM
    pizzas
GROUP BY
    pizza_type_id, -- explicit columns
    size
ORDER BY
    pizza_type_id, average_price DESC
```

- `GROUP BY ALL`

```
SELECT
    pizza_type_id,
    size,
    AVG(price) AS average_price
FROM
    pizzas
-- No need to specify columns
GROUP BY ALL
ORDER BY
    pizza_type_id, average_price DESC
```

Let's practice!

INTRODUCTION TO SNOWFLAKE