# Tables in Redshift

INTRODUCTION TO REDSHIFT

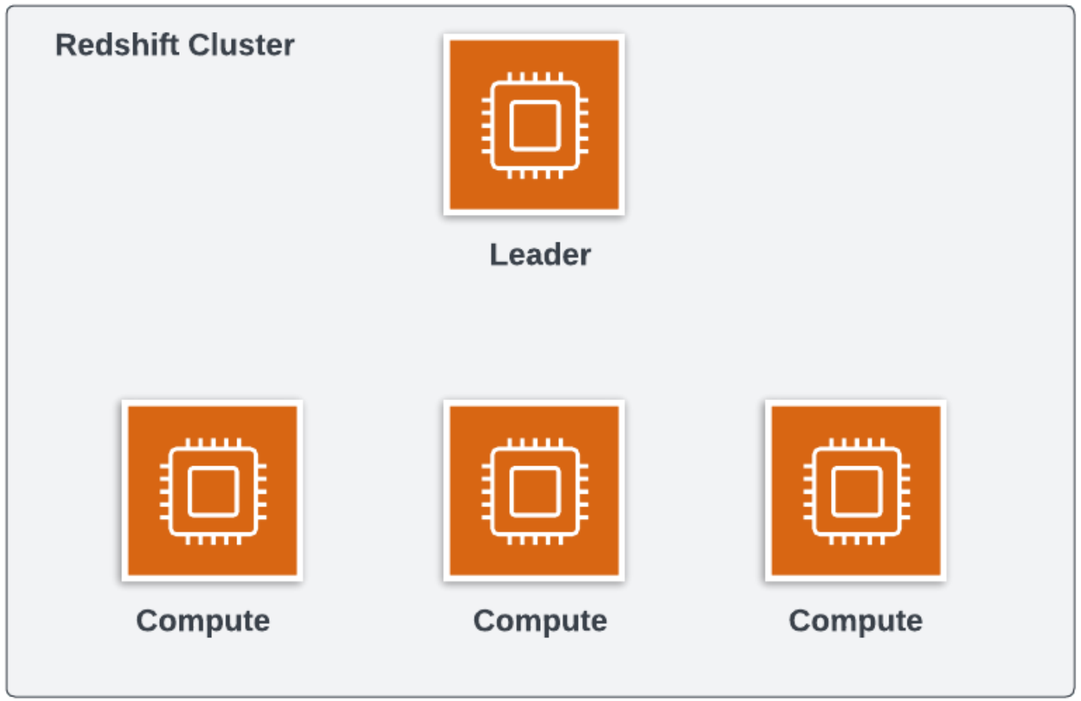
**Jason Myers**
Principal Engineer

datacamp

# Creating Tables

```
CREATE TABLE IDAHO_SITE_ID
(
    'pk_siteid' INTEGER PRIMARY KEY,
    'fk_monitoringlocation' INTEGER,
    'organization_id' VARCHAR(31),
    'organizationformalname' VARCHAR(68),
    'organization' VARCHAR(16)
);
```

# Distributing data




- Distributed across compute nodes
- Uses Redshift internal row id, `DISTKEY` , or `PRIMARY KEY`
- Several distribution styles

# Distribution styles

| Name | Description | Usage |
|---|---|---|
| ALL | Entire table on every node | Small fact lookup tables we need often in joins |
| KEY | Distributed by data in the DISTKEY column | When we aggregate or join by DISTKEY |
| EVEN | In turn distribution across nodes by row | Large tables with that don't have keys |
| AUTO | Uses ALL for small tables. Key as it grows if there are suitable DISTKEYS falling back to EVEN | default |

# DISTKEY

```
CREATE TABLE IDAHO_SITE_ID
(
    'pk_siteid' INTEGER PRIMARY KEY,
    'fk_monitoringlocation' INTEGER,
    'organization_id' VARCHAR(31),
    'organizationformalname' VARCHAR(68),
    'organization' VARCHAR(16)
)
-- Sets the data distribution key
-- to organization_id
DISTKEY(organization_id);
```

```
CREATE TABLE IDAHO_SITE_ID
(
    'pk_siteid' INTEGER PRIMARY KEY,
    'fk_monitoringlocation' INTEGER,
    -- Sets organization_id as the data
    -- distribution key
    'organization_id' VARCHAR(31) DISTKEY,
    'organizationformalname' VARCHAR(68),
    'organization' VARCHAR(16)
);
```

# Setting the distribution style

```
CREATE TABLE IDAHO_SITE_ID
(
    'pk_siteid' INTEGER PRIMARY KEY,
    'fk_monitoringlocation' INTEGER,
    'organization_id' VARCHAR(31),
    'organizationformalname' VARCHAR(68),
    'organization' VARCHAR(16)
)
DISTKEY(organization_id)
-- Sets the distribution style to key
DISTSTYLE KEY;
```

# SORTKEY

```
CREATE TABLE IDAHO_SITE_ID
(
    'pk_siteid' INTEGER PRIMARY KEY,
    'fk_monitoringlocation' INTEGER,
    'organization_id' VARCHAR(31),
    'organizationformalname' VARCHAR(68),
    'organization' VARCHAR(16)
)
DISTKEY(organization_id)
DISTSTYLE KEY
-- Sets the data sort key
-- to fk_monitoringlocation
SORTKEY(fk_monitoringlocation);
```

- Controls the storage order on disk
- Amplifies the power of predicate-pushdown
- Can have multiple

# Defining multiple SORTKEYs

```
CREATE TABLE IDAHO_SITE_ID
(
    'pk_siteid' INTEGER PRIMARY KEY,
    'fk_monitoringlocation' INTEGER,
    'organization_id' VARCHAR(31),
    'organizationformalname' VARCHAR(68),
    'organization' VARCHAR(16)
)
-- Sets fk_monitoringlocation, organization_id as compound sort keys
COMPOUND SORTKEY(fk_monitoringlocation, organization_id);
```

# Viewing column DISTKEY and SORTKEY status

```
-- View the dist and sortkey
-- status of a column
SELECT column_name,
       distkey,
       sortkey
  FROM SVV_REDSHIFT_COLUMNS
 -- Only in the spectrumdb schema
 WHERE schema_name = 'spectrumdb'
   -- For the ecommerce_sales table
   AND table_name = 'ecommerce_sales';
```

- Results

```
   column    | distkey | sortkey
=============|=========|========
year_qtr     | t       | 1
total_sales  | f       | 2
ecom_sales   | f       | 0
```

- Distkey Column
  - t for True - F for False
- SORTKEY Column
  - # is position in sort key, 0 is not part of sortkey

# Viewing distribution style

- `SVV_TABLE_INFO`
- Table details that affect query performance
  - distribution style
  - distribution skew
  - table size
  - sortkeys
  - sortkey skew

```
table          | encoded | diststyle       | sortkey1     | skew_sortkey1 | skew_rows
===============|=========|=================|==============|===============|===========
ecommerce_sales| N       | KEY(year_qtr)   | year_qtr     |               |
date           | N       | ALL             | dateid       |          1.00 |
```

# Viewing distribution style continued

```
SELECT table
       diststyle
  FROM SVV_TABLE_INFO
 WHERE schema like 'spectrumdb';
```

```
table           | diststyle
================|==============
ecommerce_sales | KEY(year_qtr)
```

# Let's practice!

INTRODUCTION TO REDSHIFT

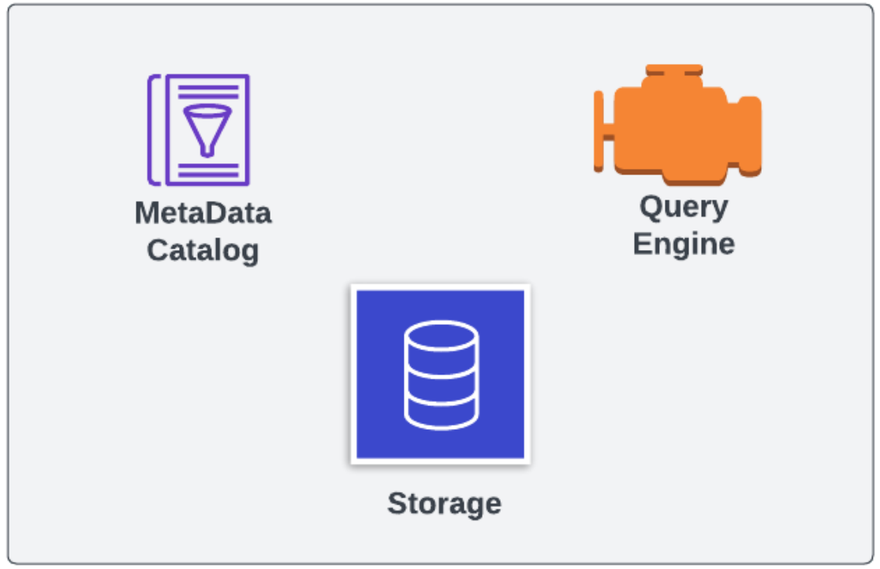
# External Schemas, File, and Table Formats

INTRODUCTION TO REDSHIFT

**Jason Myers**
Principal Architect

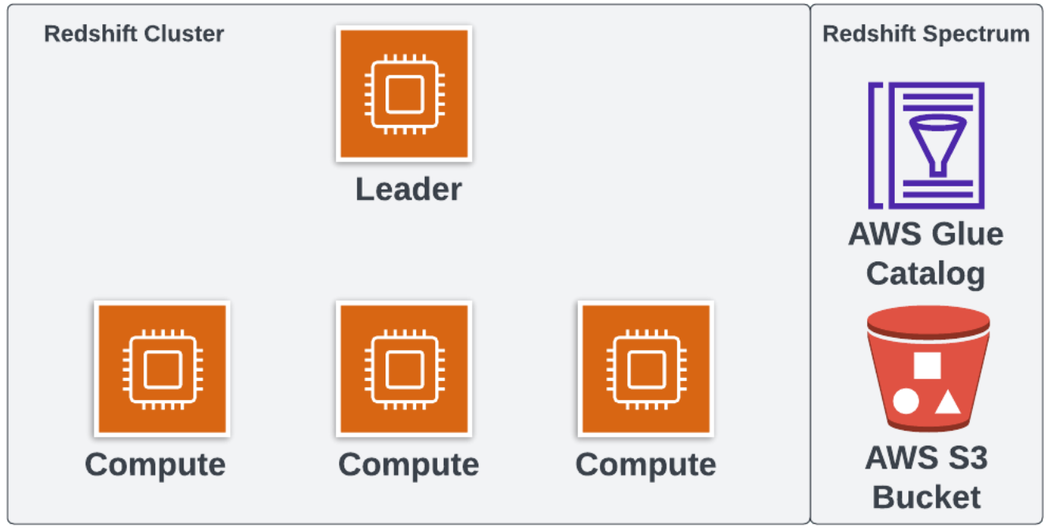
# External schemas

## Database Components




- When metadata catalog and storage are not part of the cluster it is considered external

## Redshift Spectrum




- Redshift is the engine
- Uses AWS Glue Data Catalog and AWS S3 storage by default

# S3 data file formats

| File format | Columnar | Supports parallel reads |
|---|---|---|
| Parquet | Yes | Yes |
| ORC | Yes | Yes |
| TextFile | No | No |
| OpenCSV | No | Yes |
| JSON | No | No |

# Create CSV External Table

```
CREATE TABLE spectrumdb.IDAHO_SITE_ID
(
    'pk_siteid' INTEGER PRIMARY KEY,
    -- Cutting the rest of columns for space
)
-- CSV rows are comma delimited
ROW FORMAT DELIMITED
-- CSV fields are terminated by a comma
FIELDS TERMINATED BY ','
-- CSVs are a type of text file
STORED AS TEXTFILE
-- This is where the data is in AWS S3
LOCATION 's3://spectrum-id/idaho_sites/'
-- This file has headers that we want to skip
TABLE PROPERTIES ('skip.header.line.count'='1');
```

# Querying spectrum tables

- Works just like querying internal tables
- `EXPLAIN` will look different
- No concern for `DISTKEY` or `SORTKEY` s
- Pseudocolumns
  - `$path` - shows the file storage path for the row
  - `$size` - shows the file size for the row

# Using pseudocolumns

```
SELECT "$path",
       "$size",
       pk_siteid
  FROM spectrumdb.idaho_site_id;
```

```
$path                                | $size | pk_siteid
=====================================|=======|==========
's3://spectrum-id/idaho_sites/' | 1616  | 1
's3://spectrum-id/idaho_sites/' | 1616  | 2
's3://spectrum-id/idaho_sites/' | 1616  | 3
```

# Table formats

- Common Formats:
  - Hive
  - Iceberg
  - Hudi
  - Deltalake
- Read only
- Some like Hive need an external catalog other than AWS Glue

# Viewing external schemas

- SVV_ALL_SCHEMAS - `internal` or `external`

```
SELECT schema_name,
       schema_type
  FROM SVV_ALL_SCHEMAS
 ORDER BY SCHEMA_NAME;
```

```
schema_name           | schema_type
======================|=============
public_intro_redshift | internal
spectrumdb            | external
```

# Viewing external tables

- SVV_ALL_TABLES - `TABLE` or `EXTERNAL TABLE`

```
SELECT table_name,
       table_type
  FROM SVV_ALL_TABLES
 WHERE schema_name = 'public_intro_redshift';
```

```
table_name                | table_type
==========================|================
coffee_county_weather     | TABLE
idaho_monitoring_location | TABLE
idaho_samples             | TABLE
ecommerce_sales           | EXTERNAL_TABLE
```

# Let's practice!

INTRODUCTION TO REDSHIFT

# Semistructured data

INTRODUCTION TO REDSHIFT

**Jason Myers**
Principal Architect

# Semistructured data

- Stored in Redshift's `SUPER` type
- Has dedicated SQL functions for different formats
- JSON is a type of semistructured data

# JSON Validation Functions

- `IS_VALID_JSON` - makes sure an entire JSON object is valid

```
SELECT IS_VALID_JSON('{"one":1, "two":2}');
```

```
IS_VALID_JSON
=============
true
```

```
SELECT IS_VALID_JSON('{"one":1, "two":2');
```

```
IS_VALID_JSON
=============
false
```

- IS_VALID_JSON_ARRAY - makes sure an JSON array is valid.

```
SELECT IS_VALID_JSON_ARRAY('{"one":1}')
```

```
IS_VALID_JSON_ARRAY
===================
false
```

```
SELECT IS_VALID_JSON_ARRAY('[1,2,3]')
```

```
IS_VALID_JSON_ARRAY
===================
true
```

# Extracting from a JSON object

- `JSON_EXTRACT_PATH_TEXT` returns the value at a json path
- Takes a JSON string or field then a one or more keys to the path

```
SELECT JSON_EXTRACT_PATH_TEXT('{"one":1, "two":2}', 'one');
```

```
JSON_EXTRACT_PATH_TEXT
======================
1
```

# Attempting to parse invalid JSON

```
-- Trying to extract a path from poorly formed JSON
SELECT JSON_EXTRACT_PATH_TEXT('{"one":1, "two":2', 'one');
```

```
JSON parsing error
DETAIL:
  -----------------------------------------
  error:  JSON parsing error
  code:      8001
  context:   invalid json object
  query:     [child_sequence:'one']
  location:  funcs_json.hpp:202
  process:   padbmaster [pid=1073807529]
  -----------------------------------------
```

# Extracting from nested JSON paths

- Supply multiple key arguments

```
SELECT JSON_EXTRACT_PATH_TEXT('
        {
          "one_object":{
             "nested_three": 3,
             "nested_four":4
          },
          "two":2
        }',
        'one_object', 'nested_three');
```

```
JSON_EXTRACT_PATH_TEXT
======================
3
```

# Extracting from nonexistent nested JSON paths

- Returns `NULL`

```
SELECT JSON_EXTRACT_PATH_TEXT('
       {
         "one_object":{
            "nested_three": 3,
            "nested_four":4
         },
         "two":2
       }',
       'two', 'nested_five');
```

```
JSON_EXTRACT_PATH_TEXT
======================
NULL
```

# Extracting from a JSON array

- `JSON_EXTRACT_ARRAY_ELEMENT_TEXT` returns the value at array index
- Takes a JSON string or field then an integer of the index starting from zero.

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('[1.1,400,13]', 2);
```

```
JSON_EXTRACT_ARRAY_ELEMENT_TEXT
==============================
13
```

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('[1.1,400,13]', 3);
```

```
JSON_EXTRACT_ARRAY_ELEMENT_TEXT
==============================
NULL
```

# Extracting from a nested JSON array

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT(
           JSON_EXTRACT_PATH_TEXT('
               {
                 "one":1,
                 "nested_two":[3,4,5]
               }',
               -- Extracting the value at nested_two with JSON_EXTRACT_PATH_TEXT
               'nested_two'
            ),
            -- Extracting item at position one of the array with JSON_EXTRACT_ARRAY_ELEMENT_TEXT
            1
       );
```

```
JSON_EXTRACT_ARRAY_ELEMENT_TEXT
===============================
4
```

# Extracting from a nested JSON array shortcut

- Array index has to be a string

```
-- Passing two keys ('nested_two', '1') to select
-- the second element of the nested_two keys array value
SELECT JSON_EXTRACT_PATH_TEXT('
          {
            "one":1,
            "nested_two":[3,4,5]
          }',
          'nested_two', '1'
       );
```

```
JSON_EXTRACT_PATH_TEXT
======================
4
```

# Casting in CTEs

```
WITH location_details AS (
    SELECT '{
        "location": "Salmon Challis National Forest",
      }'::SUPER::VARCHAR AS data
)
```

- Accessible as location_details.data

# Let's practice!

INTRODUCTION TO REDSHIFT