Databricks Apache Spark SQL for Data Analysts

W2 Big Data and Apache Spark

W3 Spark SQL on Databricks, Data Visualization, and Exploratory Data Analysis

W4 Spark SQL Powered Queries and Spark User Interface

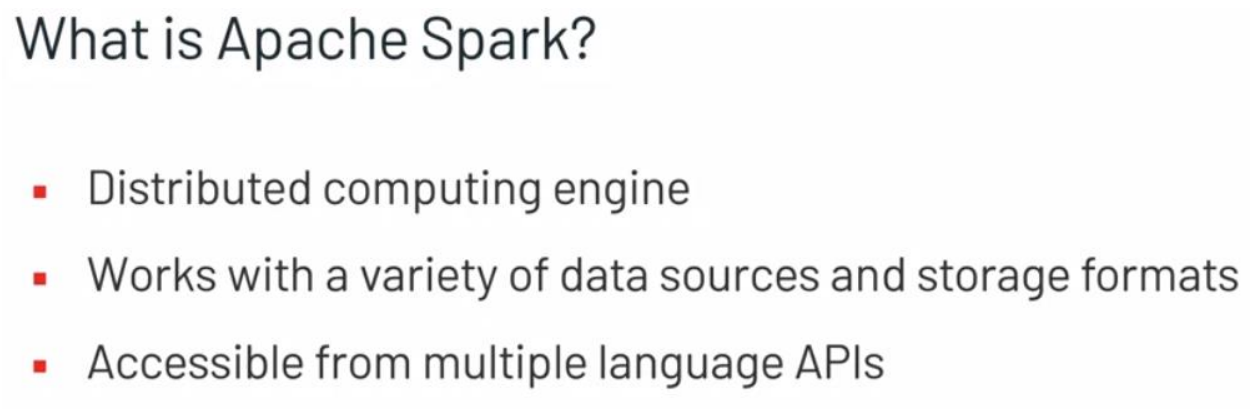
W5 Manage Nested Data Structure, Manipulating data, and Data Munging

W6 Higher Order Functions, Aggregating and Summarizing, Partitioning Tables, and Sharing Insights

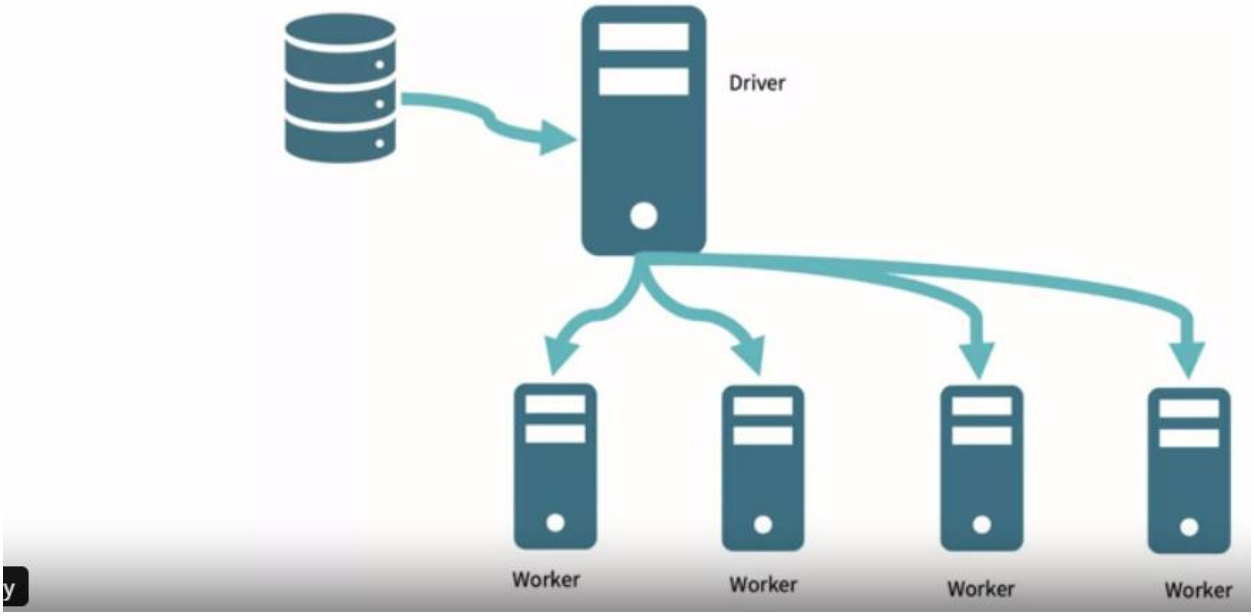
W7 Modern Data Storage and Using Delta Lake

W8 Building and Maintaining Delta Tables, Managing records in delta table, Delta Engine Optimization

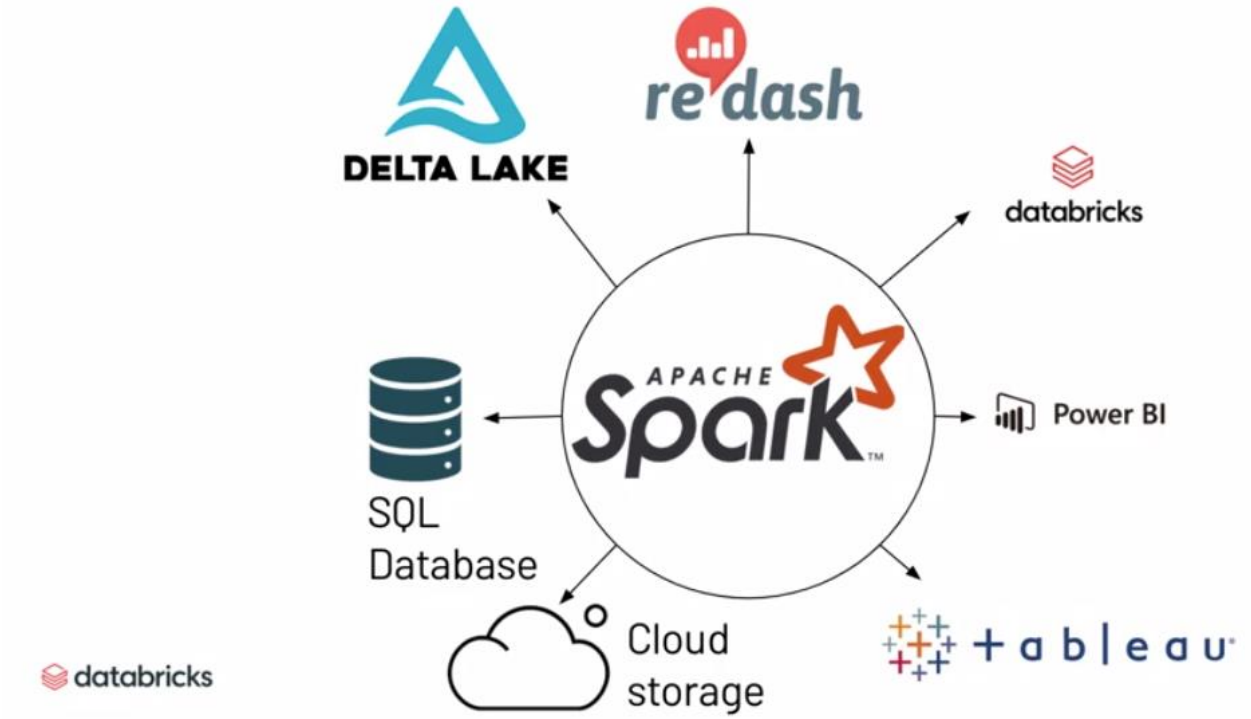
W9 SQL Coding Challenges



Distributed computing engine

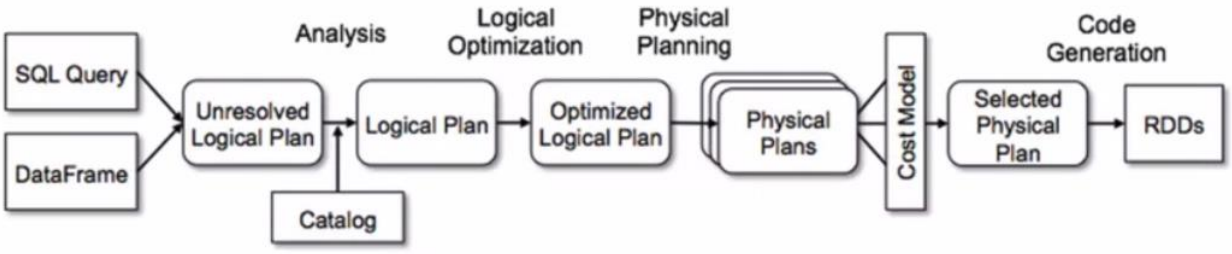


Connect to multiple data stores and BI tools



Accessible from multiple language APIs

- Access Spark from many languages:
  - SQL
  - R
  - Python
  - Scala



Why use Spark SQL?

- Ease of use
- Optimized queries
- Unified analytics teams

```
SELECT
  movieId,
  tag,
  AVG(rating)
FROM
  taggedMovieRatings
WHERE
  lower(tag) LIKE "comedy"
 GROUP BY movieId, tag;
```

3.2 Basic Queries (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revision h

Cmd 2

Basic Queries with Spark SQL

Run the following queries to start working with Spark SQL. As you work, notice that Spark SQL syntax and patterns are the same as the SQL you would use in other modern database systems.

Cmd 3

Getting Started

When you work with Databricks as part of an organization, it is likely that your workspace will be set up for you. In other words, you will be connected to various data stores and able to pull current data into your notebooks for analysis. In this course, you will use data provided by Databricks. The cell below runs a file that connects this workspace to data storage. You must run the cell below at the start of any new session. There's no need to worry about the output of this cell unless you get an error. It is simply preparing your workspace to be used with the this notebook.

Cmd 4

```
1  %run ../Includes/Classroom-Setup
```

```
Command took 13.28 seconds -- by a user at 4/5/2021, 8:46:21 AM on unknown cluster

Mounting course-specific datasets to /mnt/training...
Datasets are already mounted to /mnt/training from s3a://databricks-corp-training/common

res1: Boolean = false

res2: Boolean = false
```

Cmd 5

Create table

We are going to be working with different files (and file formats) throughout this course. The first thing we need to do, in order to access the data through our SQL interface, is create a **table** from that data.

A Databricks table is a collection of structured data. We will use Spark SQL to query tables.This table contains 10 million fictitious records that hold facts about people, like first and last names, date of birth, salary, etc. We're using the Parquet file format, which is commonly used in many big data workloads. We will talk more about various file formats later in this course. Run the code below to access the table we'll use for the first part of this lesson.

We will embed links to documentation about Databricks, Spark, and Spark SQL throughout this course. It's important to be able to read an access documentation for any system. We encourage you to get comfortable with the docs and learn more by reading through the provided links.

Cmd 6

```
1  DROP TABLE IF EXISTS People10M;
2  CREATE TABLE People10M
3  USING parquet
4  OPTIONS (
5  path "/mnt/training/dataframes/people-10m.parquet",
6  header "true");
```

▸ (1) Spark Jobs

OK

```
Command took 1.69 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:49:17 PM on My Cluster
```

Cmd 7

Querying tables

In the first part of this lesson, we 'll be using a table that has been defined for you, `People10M` . This table contains 10 million fictitious records.

We start with a simple `SELECT` statement to get a view of the data.

For now, you can think of a **table** much as you would a spreadsheet with named columns. The actual data is a file in an object store. When we define a table like the one in this exercise, it becomes available to anyone who has access to this Databricks workspace. We will view and work with this table programmatically,but you can also preview it using the `Data` tab in the sidebar on the left side of the screen.

Cmd 8

```
1  SELECT * FROM People10M;
```

▸ (1) Spark Jobs

	id	firstName	middleName	lastName	gender	birthDate	ssn	salary
1	1	Pennie	Carry	Hirschmann	F	1955-07-02T04:00:00.000+0000	981-43-9345	56172
2	2	An	Amira	Cowper	F	1992-02-08T05:00:00.000+0000	978-97-8086	40203
3	3	Quyen	Marlen	Dome	F	1970-10-11T04:00:00.000+0000	957-57-8246	53417
4	4	Coralie	Antonina	Marshal	F	1990-04-11T04:00:00.000+0000	963-39-4885	94727
5	5	Terrie	Wava	Bonar	F	1980-01-16T05:00:00.000+0000	964-49-8051	79908
6	6	Chassidy	Concepcion	Bourthouloume	F	1990-11-24T05:00:00.000+0000	954-59-9172	64652
7	7	Geri	Tambra	Mosby	F	1970-12-19T05:00:00.000+0000	968-16-4020	38195

We can view the schema for this table by using the `DESCRIBE` function.

The **schema** is a list that defines the columns in a table and the datatypes within those columns.

Cmd 12

```
1  DESCRIBE People10M;
```

	col_name	data_type	comment
1	id	int	null
2	firstName	string	null
3	middleName	string	null
4	lastName	string	null
5	gender	string	null
6	birthDate	timestamp	null
7	ssn	string	null

Showing all 8 rows.

```
Command took 0.21 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:12 PM on My Cluster
```

Cmd 13

Displaying query results

Any query that starts with a `SELECT` statement automatically displays the results below. We can use a `WHERE` clause to limit the results to those that meet a given condition or set of conditions.

For the next query, we limit the result colums to `firstName` , `middleName` , `lastName` , and `birthdate` . We use a `WHERE` clause at the end to identify that we want to limit the result set to people born after 1990 whose gender is listed as `F` .

Since `birthDate` is a timestamp type, we can extract the year of birth using the function `YEAR()`

Cmd 14

```
SELECT
  firstName,
  middleName,
  lastName,
  birthDate
FROM
  People10M
WHERE
  year(birthDate) > 1990
  AND gender = 'F'
```

▸ (1) Spark Jobs

	firstName	middleName	lastName	birthDate
1	An	Amira	Cowper	1992-02-08T05:00:00.000+0000
2	Caroyln	Mamie	Cardon	1994-05-15T04:00:00.000+0000
3	Yesenia	Eileen	Goldring	1997-07-09T04:00:00.000+0000
4	Hedwig	Dulcie	Pendleberry	1998-12-02T05:00:00.000+0000
5	Kala	Violeta	Lyfe	1994-06-23T04:00:00.000+0000
6	Gussie	India	McKeeman	1991-11-15T05:00:00.000+0000
7	Pansy	Suzie	Shrieves	1991-05-24T04:00:00.000+0000

Truncated results, showing first 1000 rows.

Command took 1.32 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:16 PM on My Cluster

Cmd 15

Math

Spark SQL includes many built-in functions that are also used in standard SQL. We can use them to create new columns based on a rule. In this case, we use a simple math function to calculate 20% of a person's listed. We use the keyword `AS` to rename the new column `savings`.

Many financial planning experts agree that 20% of a person's income should go into savings.

Cmd 16

```
SELECT
  firstName,
  lastName,
  salary,
  salary * 0.2 AS savings
FROM
  People10M
```

▸ (1) Spark Jobs

	firstName	lastName	salary	savings
1	Pennie	Hirschmann	56172	11234.4
2	An	Cowper	40203	8040.6
3	Quyen	Dome	53417	10683.4
4	Coralie	Marshal	94727	18945.4
5	Terrie	Bonar	79908	15981.6
6	Chassidy	Bourthouloume	64652	12930.4
7	Geri	Mosby	38195	7639.0

Truncated results, showing first 1000 rows.

Command took 1.08 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:23 PM on My Cluster

Cmd 17

Temporary Views

So far, you've been working with Spark SQL by querying a table that we defined for you. In the following exercises, we will work with **temporary views**. Temporary views are useful for data exploration. It gives you a name to query from SQL, but unlike a table, does not carry over when you restart the cluster or switch to a new notebook. Also, temporary views will not show up in the `Data` tab.

In the cell below, we create a temporary view that holds all the information from our last query, plus, adds another new column, `birthYear`.

```
CREATE OR REPLACE TEMPORARY VIEW PeopleSavings AS
SELECT
  firstName,
  lastName,
  year(birthDate) as birthYear,
  salary,
  salary * 0.2 AS savings
FROM
  People10M;
```

OK

Command took 0.12 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:28 PM on My Cluster

Cmd 19

Where are the results?!

When you create a temporary view, the "OK" at the bottom indicates that your command ran successfully, but the view itself does not automatically appear. To see the records in the the temporary view, you can run a query on it.

Cmd 20

```
SELECT * FROM PeopleSavings;
```

▸ (1) Spark Jobs

	firstName	lastName	birthYear	salary	savings
1	Pennie	Hirschmann	1955	56172	11234.4
2	An	Cowper	1992	40203	8040.6
3	Quyen	Dome	1970	53417	10683.4
4	Coralie	Marshal	1990	94727	18945.4
5	Terrie	Bonar	1980	79908	15981.6
6	Chassidy	Bourthouloume	1990	64652	12930.4
7	Geri	Mosby	1970	38195	7639.0

Truncated results, showing first 1000 rows.

Command took 1.31 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:34 PM on My Cluster

Query Views

For the most part, you can query a view exactly as you would query a table. The query below uses the built-in function `AVG()` to calculate `avgSalary` **grouped by** `birthYear`. This is an aggregate function, which means it's meant perform an calculation on a set of values. You must include a `GROUP BY` clause to identify the subset of values you want to summarize.

The final clause, `ORDER BY`, declares the column that will control the order in which the rows appear, and the keyword `DESC` means they will appear in descending order.

We use a `ROUND()` function around the `AVG()` to round to the nearest cent.

Cmd 22

```
SELECT
  birthYear,
  ROUND(AVG(salary), 2) AS avgSalary
FROM
  peopleSavings
GROUP BY
  birthYear
ORDER BY
  avgSalary DESC
```

▸ (2) Spark Jobs

	birthYear	avgSalary
1	2000	72741.39
2	1987	72725.18
3	1963	72722.43
4	1951	72704.04
5	1964	72693.98
6	1996	72693.55
7	1965	72675.08

Showing all 50 rows.

Command took 7.74 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:39 PM on My Cluster

Define a new table

Now we will show you how to create a table using Parquet. Parquet is an open-source, column-based file format. Apache Spark supports many different file formats; you can specify how you want your table to be written with the `USING` keyword.

For now, we will focus on the commands we will use to create a new table.

This data contains information about the relative popularity of first names in the United States by year from 1880 - 2016.

`Line 1` : Tables must have unique names. By including the `DROP TABLE IF EXISTS` command, we are ensuring that the next line (`CREATE TABLE`) can run successfully even if this table has already been created. The semi-colon at the end of the line allows us to run another command in the same cell.

`Line 2` : Creates a table named `ssaNames`, defines the data source type (`parquet`) and indicated that there are some optional parameters to follow.

`Line 3` : Identifies the path to the file in object storage

`Line 4` : Indicates that the first line of the table should be treated as a header.

Cmd 24

```
DROP TABLE IF EXISTS ssaNames;
CREATE TABLE ssaNames USING parquet OPTIONS (
  path "/mnt/training/ssn/names.parquet",
  header "true"
)
```

▸ (1) Spark Jobs

OK

Command took 0.75 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:53 PM on My Cluster

Cmd 25

Preview the data

Run the cell below to preview the data. Notice that the `LIMIT` keyword restricts the number of returned rows to the specified limit.

Cmd 26

```
SELECT
  *
FROM
  ssaNames
LIMIT
  5;
```

▸ (1) Spark Jobs

	firstName	gender	total	year
1	Jennifer	F	54336	1983
2	Jessica	F	45278	1983
3	Amanda	F	33752	1983
4	Ashley	F	33292	1983
5	Sarah	F	27228	1983

Showing all 5 rows.

Command took 1.12 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:51:57 PM on My Cluster

Cmd 27

Joining two tables

We can combine these tables to get a sense of how the data may be related. For example, you may wonder

> How many popular first names appear in our generated `People10M` dataset?

We will use a join to help answer this question. We will perform the join in a series of steps.

Cmd 28

Count distinct values

First, we query tables to get a list of the distinct values in any field. Run the commands below to see the number of distinct names are in each of our tables.

```
1  SELECT count(DISTINCT firstName)
2  FROM SSANames;
```

▸ (3) Spark Jobs

	count(DISTINCT firstName)
1	93889

Showing all 1 rows.

Command took 3.22 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:52:05 PM on My Cluster

Cmd 30

```
1  SELECT count(DISTINCT firstName)
2  FROM People10M;
```

▸ (3) Spark Jobs

	count(DISTINCT firstName)
1	5113

Showing all 1 rows.

Command took 5.34 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:52:07 PM on My Cluster

Create temporary views

Next, we create two temporary views so that the actual join will be easy to read/write.

Cmd 32

```
1  CREATE OR REPLACE TEMPORARY VIEW SSADistinctNames AS
2    SELECT DISTINCT firstName AS ssaFirstName
3    FROM SSANames;
4
5  CREATE OR REPLACE TEMPORARY VIEW PeopleDistinctNames AS
6    SELECT DISTINCT firstName
7    FROM People10M
```

OK

Command took 0.15 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:52:15 PM on My Cluster

Cmd 33

Perform join

Now, we can use the view names to **join** the two data sets. If you are new to using SQL, you may want to learn more about the different types of joins you can perform. This wikipedia article offers complete explanations, with pictures and sample SQL code.

By default, the join type shown here is `INNER`. That means the results will contain the intersection of the two sets, and any names that are not in both sets will not appear. Note, becuase it is default, we did not specify the join type.

Cmd 34

```
1  SELECT firstName
2  FROM PeopleDistinctNames
3  JOIN SSADistinctNames ON firstName = ssaFirstName
```

▸ (3) Spark Jobs

	firstName
1	Susanna
2	Julianne
3	Lashanda
4	Kiana

How many names?

To answer the question posed previously, we can perform this join and include a count of the number of records in the result.

Cmd 36

```
1  SELECT count(*)
2  FROM PeopleDistinctNames
3  JOIN SSADistinctNames ON firstName = ssaFirstName;
```

▸ (4) Spark Jobs

	count(1)
1	5096

Showing all 1 rows.

Command took 8.42 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:52:41 PM on My Cluster

Cmd 37

```
1  %run ../Includes/Classroom-Cleanup
2
```

Command took 0.04 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:52:51 PM on My Cluster

Cmd 38

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

3.3 Data Visualization (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revision his

Data Visualization with Databricks

In this lesson, you'll learn how to create and share data visualizations in Databricks.

By the end of the lesson, you will be able to:

- Create a table with a specified schema
- Cast a column as a timestamp and extract day, month, or year
- Use in-notebook visualizations to see your data

Step 1: Read through and run all the cells in this notebook.
Step 2: View the corresponding video to see instructions for visualizing and sharing data.

Cmd 3

Getting Started

Run the following cell to connect your workspace to the appropriate data source.

Cmd 4

```
1 %run ../Includes/Classroom-Setup
```

Command took 53.20 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 10:36:59 PM on My Cluster

Mounting course-specific datasets to **/mnt/training**...
Mounted datasets to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false

res2: Boolean = false
```

Cmd 5

Create a Table

In the previous lesson, we created a table for you to start querying. In this lesson, you will create the table by reading directly from the data source and specifying a **schema**. A schema describes the structure of your data. It contains column names and the type of data in each column. All tables must have an associated schema; if you do not explicitly define one, Spark may be able to infer it.

In the cell below, we define the schema as we create the table. This data has the following schema:

Column Name	Type
userId	INT
movieId	INT
rating	FLOAT
timeRecorded	INT

Notice that it is defined right after the `CREATE TABLE` statement with the name of each column followed by the datatypes within the column. The whole group of columns is surround by parentheses and each individual column is spereated by a comma.

Cmd 6

```
1  DROP TABLE IF EXISTS movieRatings;
2  CREATE TABLE movieRatings (
3    userId INT,
4    movieId INT,
5    rating FLOAT,
6    timeRecorded INT
7  ) USING csv OPTIONS (
8    PATH "/mnt/training/movies/20m/ratings.csv",
9    header "true"
10 );
```

OK

Command took 3.22 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 10:38:53 PM on My Cluster

Cmd 7

Preview the data

This table contains a little more than 20 million records of movies ratings submitted by users. Note that the timestamp is an integer value recorded in UTC time.

Cmd 8

```
1 SELECT
2   *
3 FROM
4   movieRatings;
```

	userId	movieId	rating	timeRecorded
1	1	2	3.5	1112486027
2	1	29	3.5	1112484676
3	1	32	3.5	1112484819
4	1	47	3.5	1112484727
5	1	50	3.5	1112484580
6	1	112	3.5	1094785740
7	1	151	4	1094785734

Truncated results, showing first 1000 rows.

Command took 3.63 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 10:38:58 PM on My Cluster

Cmd 9

Cast as timestamp

We use the `CAST()` function to show the timestamp as a human-readable time and date.

Cmd 10

```
1  SELECT
2    rating,
3    CAST(timeRecorded as timestamp)
4  FROM
5    movieRatings;
```

▸ (1) Spark Jobs

	rating	timeRecorded
1	3.5	2005-04-02T23:53:47.000+0000
2	3.5	2005-04-02T23:31:16.000+0000
3	3.5	2005-04-02T23:33:39.000+0000
4	3.5	2005-04-02T23:32:07.000+0000
5	3.5	2005-04-02T23:29:40.000+0000
6	3.5	2004-09-10T03:09:00.000+0000
7	4	2004-09-10T03:08:54.000+0000

Truncated results, showing first 1000 rows.

Create temporary view

We will create a temporary view that we can easily refer to the data we want to include in our visualization. For this data, we can investigate whether there are any patterns in the ratings when grouped by month. To do that, we use the `ROUND()` and `AVG()` functions to calculate the average rating and limit it to 3 decimal places. Then, extract the month from the `timeRecorded` column after casting it as a timestamp. The `AVG()` is calculated over the course of a month, as specified in the `GROUP BY` clause.

Cmd 12

```
CREATE
OR REPLACE TEMPORARY VIEW ratingsByMonth AS
SELECT
  ROUND(AVG(rating), 3) AS avgRating,
  month(CAST(timeRecorded as timestamp)) AS month
FROM
  movieRatings
GROUP BY
  month;
```

OK

Command took 0.44 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 10:39:22 PM on My Cluster

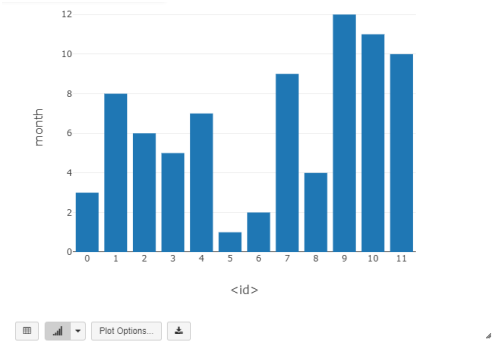
Cmd 13

Visualize the data

Run the next cell to see the view we just defined ordered by `avgRating` from least to greatest. The results will appear as a table. In the next section, you will receive video instruction that will show you how to display this table as a chart.

Cmd 14

```
SELECT
  *
FROM
  ratingsByMonth
ORDER BY
  avgRating;
```



Command took 39.43 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 10:39:27 PM on My Cluster

Cmd 15

```
%run ../Includes/Classroom-Cleanup
```

Command took 0.04 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 10:42:24 PM on My Cluster

Cmd 16

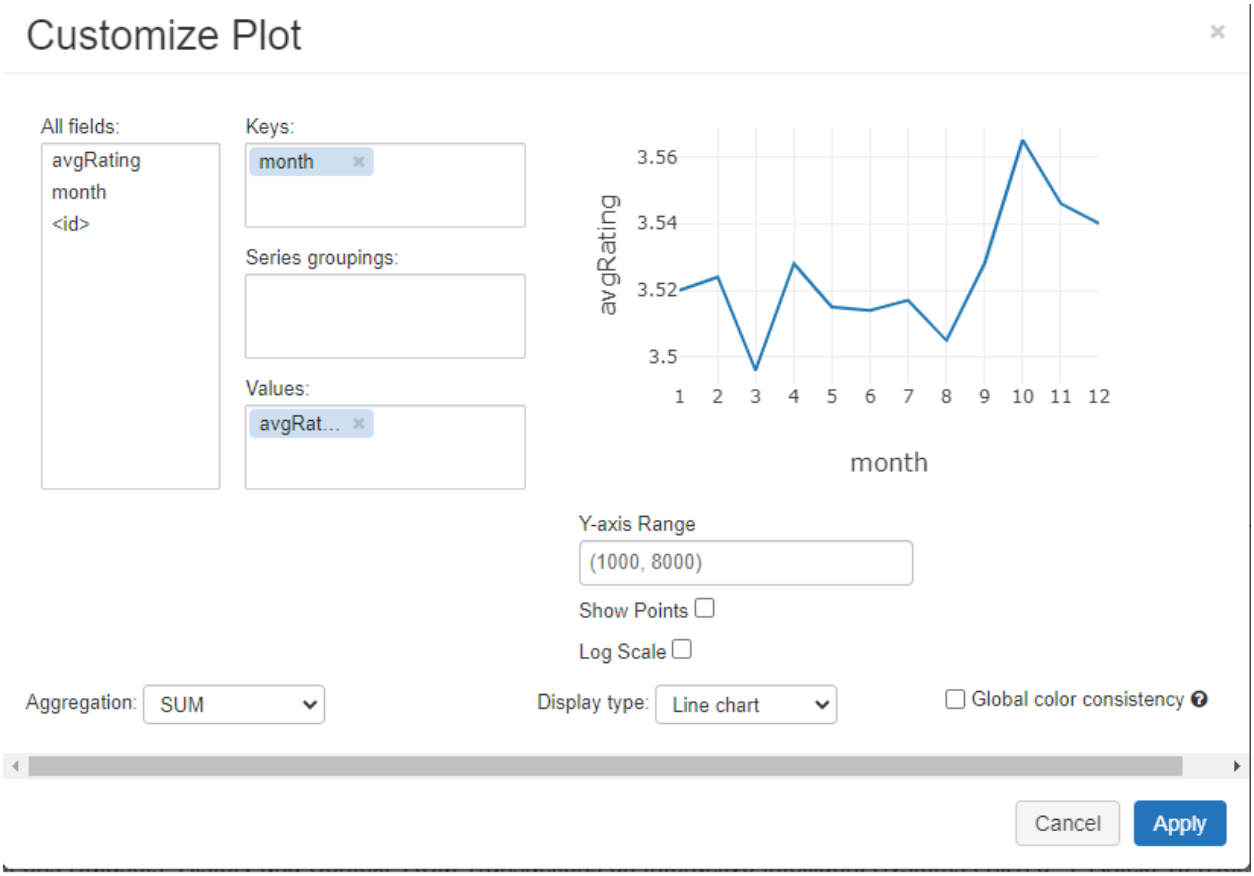
Citation

Access original data and backround information here:

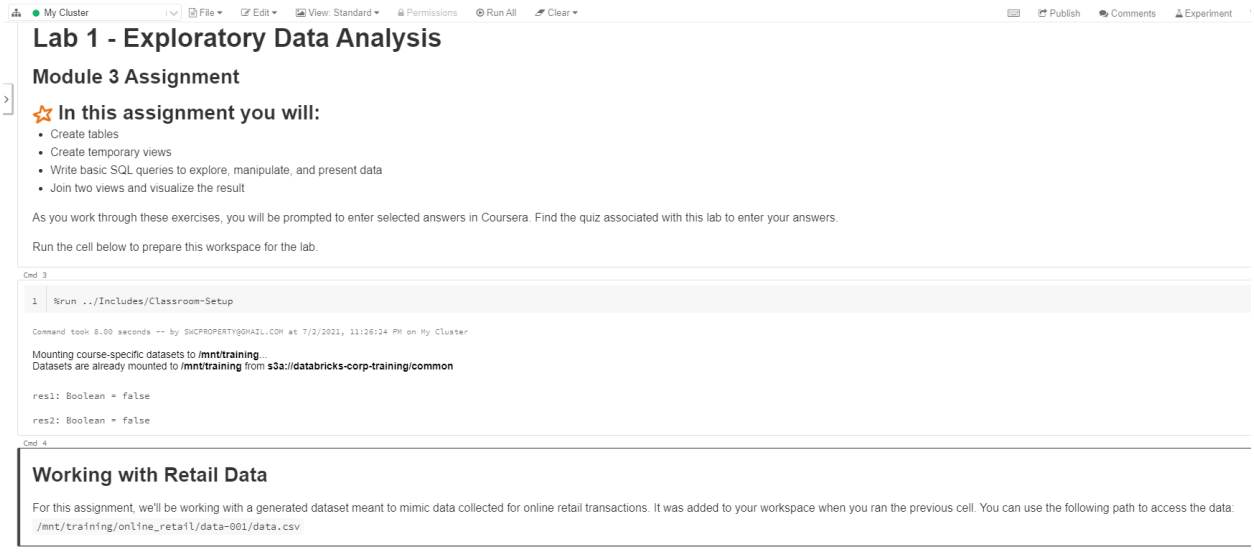
F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

Cmd 17

© 2020 Databricks, Inc. All rights reserved.



3.4 Lab: Exploratory Data Analysis (SQL)



Lab 1 - Exploratory Data Analysis

Module 3 Assignment

In this assignment you will:

- Create tables
- Create temporary views
- Write basic SQL queries to explore, manipulate, and present data
- Join two views and visualize the result

As you work through these exercises, you will be prompted to enter selected answers in Coursera. Find the quiz associated with this lab to enter your answers.

Run the cell below to prepare this workspace for the lab.

```
Cmd 3

1  %run ../Includes/Classroom-Setup

Command took 8.00 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:26:24 PM on My Cluster

Mounting course-specific datasets to /mnt/training...
Datasets are already mounted to /mnt/training from s3a://databricks-corp-training/common

res1: Boolean = false

res2: Boolean = false
```

Cmd 4

Working with Retail Data

For this assignment, we'll be working with a generated dataset meant to mimic data collected for online retail transactions. It was added to your workspace when you ran the previous cell. You can use the following path to access the data:
`/mnt/training/online_retail/data-001/data.csv`

Exercise 1: Create a table

Summary: Create a new table named `outdoorProducts` with the following schema:

Column Name	Type
invoiceNo	STRING
stockCode	STRING
description	STRING
quantity	INT
invoiceDate	STRING
unitPrice	DOUBLE
customerID	INT
countryName	STRING

Steps to complete:

- Make sure this notebook is idempotent by dropping any tables that have the name `outdoorProducts`
- Use `csv` as the specified data source
- Use the path provided above to access the data
- This data contains a header; include that in your table creation statement

```
-- TODO
Drop table if exists outdoorProducts;
create table outdoorProducts (
  invoiceNo string,
  stockCode string,
  description string,
  quantity int,
  invoiceDate string,
  unitPrice double,
  customerId int,
  countryName string
) using csv options (
  path "/mnt/training/online_retail/data-001/data.csv",
  header "true"
);
```

OK

Command took 0.76 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:31:25 PM on My Cluster

Cmd 7

Exercise 2: Explore the data

Summary: Count the number of items that have a negative `quantity`

This table keeps track of online transactions, including returns. Some of the quantities in the `quantity` column show a negative number. Run a query that counts then number of negative values in the `quantity` column.

Steps to complete:

- Write a query that reports the number of values less than 0 in the `quantity` column
- **Report the answer in the corresponding quiz in Coursera**

Cmd 8

```
-- TODO
select count(*)
from outdoorProducts
where quantity < 0;
```

	count(1)
1	1192

Showing all 1 rows.

Command took 2.12 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:32:38 PM on My Cluster

Cmd 9

Exercise 3: Create a temporary view

Summary: Create a temporary view that includes only the specified columns and rows, and uses math to create a new column.

Steps to complete:

- Create a temporary view named `sales`
- Create a new column, `totalAmount`, by multiplying `quantity` times `unitPrice` and rounding to the nearest cent
- Include columns: `stockCode`, `quantity`, `unitPrice`, `totalAmount`, `countryName`
- Include only rows where `quantity` is greater than 0

Cmd 10

```
1  -- TODO
2  create or replace temporary view sales as (
3  select stockCode, quantity, unitPrice, round(quantity * unitPrice, 2) as totalAmount, countryName
4  from outdoorProducts
5  where quantity > 0
6  );
```

OK

Command took 0.13 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:38:54 PM on My Cluster

Exercise 4: Display ordered view

Summary: Show the view you created with `totalAmount` sorted greatest to least

Steps to complete:

- Select all columns form the view `sales`
- Order the `totalAmount` column from greatest to least
- **Report the `countryName` from the row with the greatest `totalAmount` of sales in the corresponding answer area in Coursera**

Cmd 12

```
1  -- TODO
2  select * from sales
3  order by totalAmount desc
```

▸ (1) Spark Jobs

	stockCode	quantity	unitPrice	totalAmount	countryName
1	23166	74215	1.04	77183.6	United Kingdom
2	AMAZONFEE	1	13541.33	13541.33	United Kingdom
3	21108	3114	2.1	6539.4	United Kingdom
4	85123A	1930	2.55	4921.5	United Kingdom
5	48185	670	6.75	4522.5	United Kingdom
6	22470	1284	3.21	4121.64	United Kingdom
7	21623	600	6.38	3828	United Kingdom

Truncated results, showing first 1000 rows.

Command took 2.12 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:39:29 PM on My Cluster

Exercise 5: View countries

Summary: Show a list of all unique `countryName` values in the `sales` view

Steps to complete:

- Write a query that returns only distinct `countryName` values
- **Answer the corresponding question in Coursera**

Cmd 14

```
1  -- TODO
2  select distinct countryName
3  from sales
```

▸ (2) Spark Jobs

	countryName
1	Sweden
2	Germany
3	France
4	Belgium
5	Finland
6	Italy
7	EIRE

Showing all 24 rows.

Command took 2.06 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:40:34 PM on My Cluster

Exercise 6: Create a temporary view: `salesQuants`

Summary: Create a temporary view that shows total `quantity` of items purchased from each `countryName`

Steps to complete:

- Create a temporary view named `salesQuants`
- Display the sum of all `quantity` values grouped by `countryName`. Name that column `totalQuantity`
- Order the view by `totalQuantity` from greatest to least
- **Answer the corresponding question in Coursera**

Cmd 16

```
-- TODO
create or replace temporary view salesQuants as (
select countryName, sum(quantity) as totalQuantity
from sales
group by countryName
order by totalQuantity desc
);
select * from salesQuants;
```

▸ (2) Spark Jobs

	countryName	totalQuantity
1	United Kingdom	558987
2	Netherlands	21381
3	EIRE	13682
4	France	11580
5	Germany	11436
6	Australia	6001
7	Sweden	4247

Showing all 24 rows.

Command took 2.25 seconds -- by SWCPROPERTY@GMAIL.COM at 7/2/2021, 11:43:20 PM on My Cluster

Exercise 7: Read in a new parquet table

Summary: Create a new table named `countryCodes`.

Steps to complete:

- Drop any existing tables named `countryCodes` from your database
- Use this path: `/mnt/training/countries/ISOCountryCodes/ISOCountryLookup.parquet` to create a new table using parquet as the data source. Name it `countryCodes`
- Include options to indicate that there **is** a header for this table

Cmd 18

```
-- TODO
drop table if exists countryCodes;
create table countryCodes
using parquet options (
  path "/mnt/training/countries/ISOCountryCodes/ISOCountryLookup.parquet",
  header "true"
);
select * from countryCodes;
```

▸ (2) Spark Jobs

	English ShortName	alpha2Code	alpha3Code	numericCode	ISO31662SubdivisionCode	independentTerritory
1	Afghanistan	AF	AFG	004	ISO 3166-2:AF	Yes
2	Åland Islands	AX	ALA	248	ISO 3166-2:AX	No
3	Albania	AL	ALB	008	ISO 3166-2:AL	Yes
4	Algeria	DZ	DZA	012	ISO 3166-2:DZ	Yes
5	American Samoa	AS	ASM	016	ISO 3166-2:AS	No
6	Andorra	AD	AND	020	ISO 3166-2:AD	Yes
7	Angola	AO	AGO	024	ISO 3166-2:AO	Yes

Showing all 249 rows.

Command took 4.39 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 12:11:05 AM on My Cluster

Exercise 8: View metadata

Summary: View column names and data types in this table.

Steps to complete:

- Use the `DESCRIBE` command to display all of column names and their data types
- **Answer the corresponding question in Coursera**

Cmd 20

```
1  -- TODO
2  describe countryCodes;
```

	col_name	data_type	comment
1	EnglishShortName	string	null
2	alpha2Code	string	null
3	alpha3Code	string	null
4	numericCode	string	null
5	ISO31662SubdivisionCode	string	null
6	independentTerritory	string	null

Showing all 6 rows.

Command took 0.15 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 12:03:06 AM on My Cluster

Exercise 9: Join and Visualize

Summary: Use the `salesQuants` view and the `countryCodes` table to display a pie chart that shows total sales by country, and identifies the country by its 3-letter id.

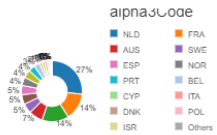
Steps to complete:

- Write a query that results in two columns: `totalQuantity` from `salesQuants` and `alpha3Code` from `countryCodes`
- Join `countryCodes` with `salesQuants` on the name of country listed in each table
- Visualize your results as a pie chart that shows the percent of sales from each country

Cmd 22

```
--TODO
select alpha3Code, totalQuantity
from salesQuants as a
inner join countryCodes as b
on a.countryName = b.EnglishShortName
```

▸ (2) Spark Jobs



Plot Options...

Command took 2.17 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 12:11:41 AM on My Cluster

Sanity Check

It's always smart to do a sanity check when manipulating and joining datasets.

- Compare your chart to the table you displayed in task #4
- Try the challenge problem to figure out what may have gone wrong

Cmd 24

Challenge: Find the problem

Click here for the next hint

Hint:

Display a distinct list of all countries in the countryCodes table. Compare that list with the distinct list of countries from the outdoorProducts chart.

Cmd 25

```
-- TODO
```

Cmd 26

```
-- TODO
```

Cmd 27

```
CREATE
OR REPLACE TEMPORARY VIEW modCountryCodes AS
SELECT
  alpha3code,
  REPLACE (
    EnglishShortName,
    "United Kingdom of Great Britain and Northern Ireland",
    "United Kingdom"
  ) AS EnglishShortName
FROM
  countryCodes;
```

OK

Command took 0.11 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 12:12:58 AM on My Cluster

```sql
-- TODO
select *
from modCountryCodes
where EnglishShortName = 'United Kingdom'
```

▸ (1) Spark Jobs

	alpha3code	EnglishShortName
1	GBR	United Kingdom

Showing all 1 rows.

Command took 0.62 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 12:13:32 AM on My Cluster

Cmd 29

```
%run ../Includes/Classroom-Cleanup

```

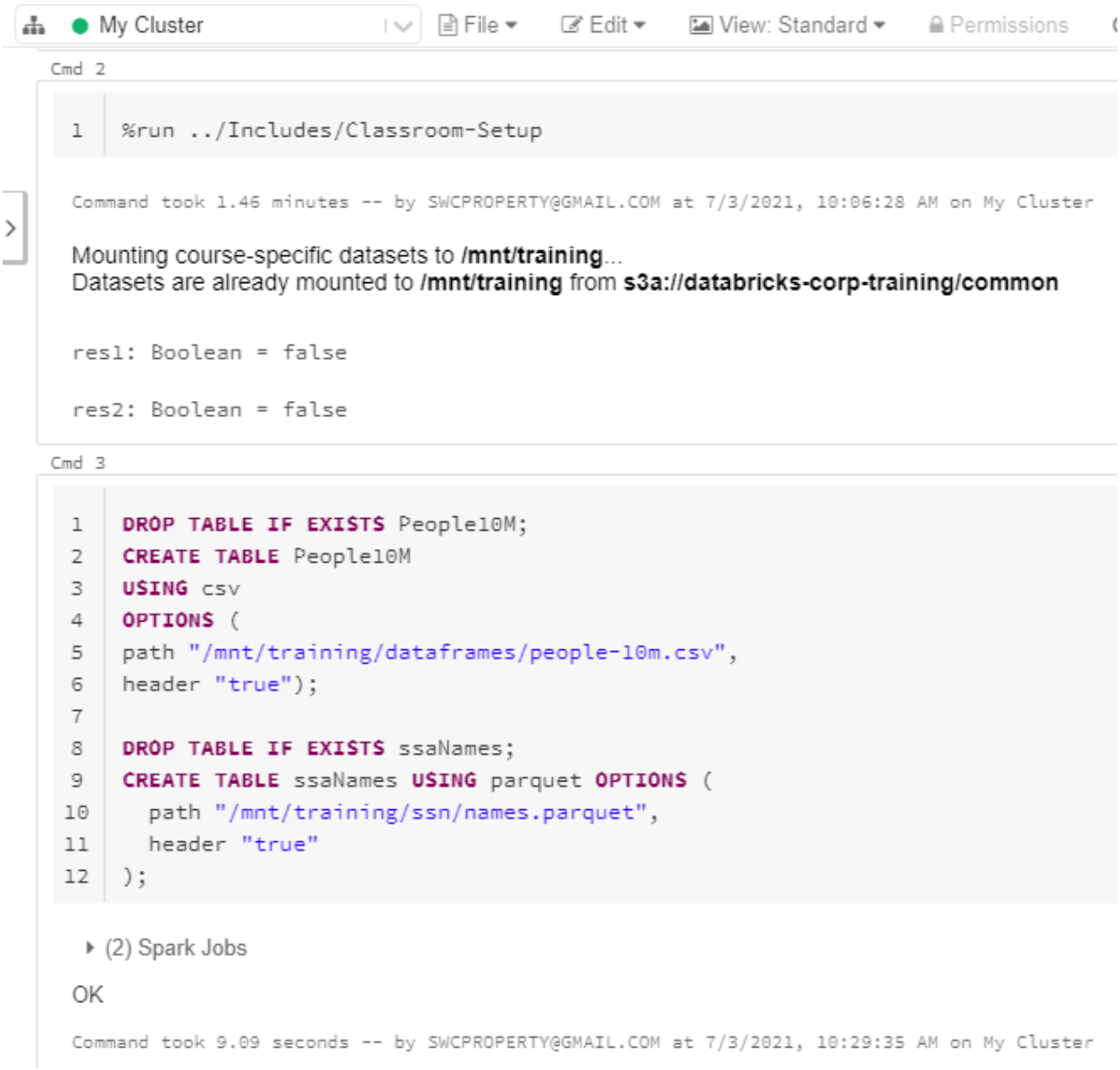
Command took 0.04 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 12:13:55 AM on My Cluster

Cmd 30

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

4.1 The Spark UI (SQL)



```sql
SELECT
  firstName,
  lastName,
  birthDate
FROM
  People10M
WHERE
  year(birthDate) > 1990
  AND gender = 'F'
```

▸ (1) Spark Jobs

	firstName	lastName	birthDate
1	An	Cowper	1992-02-08T05:00:00.000Z
2	Caroyln	Cardon	1994-05-15T04:00:00.000Z
3	Yesenia	Goldring	1997-07-09T04:00:00.000Z
4	Hedwig	Pendleberry	1998-12-02T05:00:00.000Z
5	Kala	Lyfe	1994-06-23T04:00:00.000Z
6	Gussie	McKeeman	1991-11-15T05:00:00.000Z
7	Pansy	Shrieves	1991-05-24T04:00:00.000Z

Truncated results, showing first 1000 rows.

Command took 3.36 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:29:59 AM on My Cluster

Plan Optimization Example

Cmd 7

```
1  CREATE OR REPLACE TEMPORARY VIEW joined AS
2  SELECT People10m.firstName,
3    to_date(birthDate) AS date
4  FROM People10m
5    JOIN ssaNames ON People10m.firstName = ssaNames.firstName;
6
7  CREATE OR REPLACE TEMPORARY VIEW filtered AS
8  SELECT firstName,count(firstName)
9  FROM joined
10 WHERE
11   date >= "1980-01-01"
12 GROUP BY
13   firstName, date;
14
```

OK

Command took 1.21 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:30:07 AM on My Cluster

Cmd 8

```
1  SELECT * FROM  filtered;
```

▸ (2) Spark Jobs

	firstName	count(firstName)
1	Ellan	49
2	Charline	117
3	Latisha	72
4	Tonita	73
5	Gwenn	76
6	Nidia	67
7	Torri	91

Truncated results, showing first 1000 rows.

Command took 2.76 minutes -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:30:13 AM on My Cluster

```
1  CACHE TABLE filtered;
```

▸ (2) Spark Jobs

OK

Command took 2.52 minutes -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:31:48 AM on My Cluster

Cmd 10

```
1  SELECT * FROM filtered;
```

▸ (1) Spark Jobs

	firstName	count(firstName)
1	Ellan	49
2	Charline	117
3	Latisha	72
4	Tonita	73
5	Gwenn	76
6	Nidia	67
7	Torri	91

Truncated results, showing first 1000 rows.

Command took 0.54 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:35:42 AM on My Cluster

Cmd 11

```
1  SELECT * FROM filtered WHERE firstName = "Latisha";
```

▸ (5) Spark Jobs

	firstName	count(firstName)
1	Latisha	72
2	Latisha	72
3	Latisha	72
4	Latisha	72
5	Latisha	72
6	Latisha	72
7	Latisha	72

Showing all 513 rows.

```
1  UNCACHE TABLE IF EXISTS filtered;
```

OK

Command took 1.23 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:35:52 AM on My Cluster

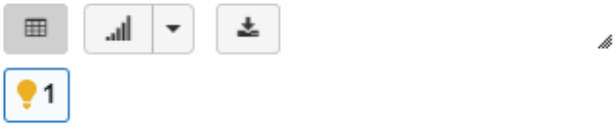
Cmd 13

```
1  SELECT * FROM filtered WHERE firstName = "Latisha";
```

▸ (2) Spark Jobs

	firstName	count(firstName)
1	Latisha	72
2	Latisha	72
3	Latisha	72
4	Latisha	72
5	Latisha	72
6	Latisha	72
7	Latisha	72

Showing all 513 rows.



Command took 22.39 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:35:55 AM on My Cluster

Cmd 14

Set Partitions

Cmd 15

```
1  DROP TABLE IF EXISTS bikeShare;
2  CREATE TABLE bikeShare
3  USING csv
4  OPTIONS (
5    path "/mnt/training/bikeSharing/data-001/hour.csv",
6    header "true")
```

Command took 1.32 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:36:03 AM on My Cluster

Cmd 16

```
SELECT
  *
FROM
  bikeShare
WHERE
  hr = 10
```

▸ (1) Spark Jobs

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	11	2011-01-01	1	0	1	10	0	6	0	1	0.38	0.3939	0.76	0.2537	12	24	36
2	34	2011-01-02	1	0	1	10	0	0	0	2	0.36	0.3485	0.81	0.2239	7	46	53
3	56	2011-01-03	1	0	1	10	0	1	1	1	0.18	0.1667	0.43	0.2537	11	33	44
4	79	2011-01-04	1	0	1	10	0	2	1	2	0.16	0.1364	0.69	0.3284	5	37	42
5	102	2011-01-05	1	0	1	10	0	3	1	1	0.22	0.197	0.37	0.3284	4	53	57
6	125	2011-01-06	1	0	1	10	0	4	1	1	0.2	0.2576	0.47	0	3	42	45
7	148	2011-01-07	1	0	1	10	0	5	1	1	0.22	0.197	0.37	0.3284	16	47	63

Showing all 727 rows.

Command took 1.15 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:36:17 AM on My Cluster

```
DROP TABLE IF EXISTS bikeShare_partitioned;
CREATE TABLE bikeShare_partitioned
PARTITIONED BY (p_hr)
  AS
SELECT
  instant,
  dteday,
  season,
  yr,
  mnth,
  hr as p_hr,
  holiday,
  weekday,
  workingday,
  weathersit,
  temp
FROM
  bikeShare
```

▸ (4) Spark Jobs

Query returned no results

Command took 24.64 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:36:27 AM on My Cluster

Cmd 18

```
SELECT * FROM bikeShare_partitioned WHERE p_hr = 10
```

▸ (1) Spark Jobs

	instant	dteday	season	yr	mnth	p_hr	holiday	weekday	workingday	weathersit	temp
1	11	2011-01-01	1	0	1	10	0	6	0	1	0.38
2	34	2011-01-02	1	0	1	10	0	0	0	2	0.36
3	56	2011-01-03	1	0	1	10	0	1	1	1	0.18
4	79	2011-01-04	1	0	1	10	0	2	1	2	0.16
5	102	2011-01-05	1	0	1	10	0	3	1	1	0.22
6	125	2011-01-06	1	0	1	10	0	4	1	1	0.2
7	148	2011-01-07	1	0	1	10	0	5	1	1	0.22

Showing all 727 rows.

Command took 1.47 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:36:30 AM on My Cluster

Beware of small files!

Cmd 20

```
DROP TABLE IF EXISTS bikeShare_parquet;
CREATE TABLE bikeShare
PARTITIONED BY (p_instant)
  AS
SELECT
  instant AS p_instant,
  dteday,
  season,
  yr,
  mnth,
  hr,
  holiday,
  weekday,
  workingday,
  weathersit,
  temp
FROM
  bikeShare_csv
```

Cmd 21

```
%run ../Includes/Classroom-Cleanup
```

```
Command took 0.05 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 10:37:00 AM on My Cluster
```

Cmd 22

Citations

Bike Sharing Data

[1] Fanaee-T, Hadi, and Gama, Joao, Event labeling combining ensemble detectors and background knowledge, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg, doi:10.1007/s13748-013-0040-3.

@article{ year={2013}, issn={2192-6352}, journal={Progress in Artificial Intelligence}, doi={10.1007/s13748-013-0040-3}, title={Event labeling combining ensemble detectors and background knowledge}, url={http://dx.doi.org/10.1007/s13748-013-0040-3}, publisher={Springer Berlin Heidelberg}, keywords={Event labeling; Event detection; Ensemble learning; Background knowledge}, author={Fanaee-T, Hadi and Gama, Joao}, pages={1-15} }

Managing Nested Data with Spark SQL

In this notebook, you'll be digging into mock data from a group of data centers. A **data center** is a dedicated space where computing and networking equipment is set up to collect, store, process, and distribute data. The continuous operation of centers like this can be crucial to maintaining continuity in business, so environmental conditions must be closely monitored.

This example uses mock data from 4 different data centers, each with four different kinds of sensors that periodically collect temperature and CO_2 level readings. Temperature and CO_2 levels are stored as arrays where temperature is collected 12 times per day and CO_2 level is collected 6 times per day.

Run the following queries to learn about how to work with and manage nested data in Spark SQL.
In this notebook, you will:

- Work with hierarchical data
- Use common table expressions (CTE)
- Create new tables based on CTEs
- Use `EXPLODE` to manage nested objects

Cmd 3

Getting started

Run the cell below to set up your classroom environment.

Cmd 4

```
%run ../Includes/Classroom-Setup
```

```
Command took 1.41 minutes -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:07:39 PM on My Cluster
```

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false

res2: Boolean = false
```

Create table

The Databricks File System (DBFS) is a distributed file system mounted into a Databricks workspace and available on Databricks clusters. In practice, this will allow you to access data that has been mounted to your workspace and interact with that storage using directories and file names instead of storage urls. In this lesson, we'll use data from datasets in object storage that has been mounted to the DBFS. We will create a table and explore some of the optional arguments available to us.

The cell below begins with a `DROP TABLE IF EXISTS` command. This means that if a table by the given name exists, it will be dropped. If it does not exist, this command does nothing. This will keep our notebook **idempotent**, meaning it could be run more than once without throwing errors or introducing extra files.

Cmd 6

```
DROP TABLE IF EXISTS DCDataRaw;
CREATE TABLE DCDataRaw
USING parquet
OPTIONS (
    PATH "/mnt/training/iot-devices/data-centers/2019-q2-q3"
    )
```

▸ (1) Spark Jobs

OK

```
Command took 5.71 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:14:06 PM on My Cluster
```

Cmd 7

View metadata and "Detailed Table Information"

In a previous lesson, we used the `DESCRIBE` command to view metadata. Run the command below to see the output when we attach the optional keyword `EXTENDED`.

You can find the same information about the schema at the top. Notice that one of our columns contains a `MapType` column, and, within that, a `StructType` field. When working with structured data, like parquet files, and semi-structured data, like JSON files, you will frequently encounter complex data types, like `MapType`, `StructType`, and `ArrayType`.

In this example, the `MapType` column holds a JSON object that has a `string` as its **key** and a `struct` field as the **value**. As you work through this notebook, we will unnest and explore that data. Learn more about the data types you will be working with in Spark SQL in the associated docs.

Detailed Table Information contains information about the table's database name, original source file type and location, and more.

```
DESCRIBE EXTENDED DCDataRaw;
```

	col_name	data_type	comment
1	dc_id	string	null
2	date	string	null
3	source	map<string,struct<description:string,ip:string,id:int,temps:array<int>,co2_level:array<int>>>	null
4			
5	# Detailed Table Information		
6	Database	default	
7	Table	dcdataraw	

Showing all 17 rows.

Command took 0.70 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:15:44 PM on My Cluster

Cmd 9

View a sample

It may you help understand the data if we view a few rows. Instead of simply returning the top rows, we can get a random sampling of rows using the function `RAND()` to return random rows and the `LIMIT` keyword to set the number of rows we want to see.

```
SELECT * FROM DCDataRaw
ORDER BY RAND()
LIMIT 3;
```

▸ (1) Spark Jobs

	dc_id	date	source
1	dc-103	2019-07-02	▸ {"sensor-igauge": {"description": "Sensor attached to the container ceilings", "ip": "262.92.152.247", "id": 14, "temps": [22, 21, 20, 25, 30, 16, 17, 18, 18, 22, 15, 19], "co2_level": [1214, 1243, 1460, 1378, 1325, 1495]}, "sensor-ipad": {"description": "Sensor ipad attached to carbon cylinders", "ip": "212.237.231.298", "id": 21, "temps": [19, 9, 11, 17, 17, 20, 26, 15, 16, 15, 11, 20], "co2_level": [1484, 1520, 1324, 1428, 1505, 1503]}, "sensor-inest": {"description": "Sensor attached to the factory ceilings", "ip": "26.105.47.133", "id": 26, "temps": [12, 20, 19, 23, 11, 22, 15, 15, 19, 17, 16, 17], "co2_level": [1397, 1398, 1474, 1301, 1584, 1389]}, "sensor-istick": {"description": "Sensor embedded in exhaust pipes in the ceilings", "ip": "139.219.212.128", "id": 29, "temps": [6, 5, 11, 2, 16, 8, 12, 7, 2, 21, 5, 7], "co2_level": [1344, 1330, 1297, 1234, 1406, 1361]}}
2	dc-101	2019-10-27	▸ {"sensor-igauge": {"description": "Sensor attached to the container ceilings", "ip": "240.259.282.215", "id": 16, "temps": [26, 14, 16, 14, 6, 15, 18, 10, 14, 15, 14, 16], "co2_level": [1279, 1200, 1193, 1350, 1008, 1286]}, "sensor-ipad": {"description": "Sensor ipad attached to carbon cylinders", "ip": "252.162.23.206", "id": 22, "temps": [13, 17, 9, 16, 16, 13, 19, 16, 18, 15, 18, 12], "co2_level": [957, 1123, 1067, 890, 988, 1044]}, "sensor-inest": {"description": "Sensor attached to the factory ceilings", "ip": "27.96.32.198", "id": 25, "temps": [8, 14, 9, 12, 18, 13, 9, 9, 9, 9, 8, 15], "co2_level": [1295, 908, 1324, 1076, 1192, 1265]}, "sensor-istick": {"description": "Sensor embedded in exhaust pipes in the ceilings", "ip": "290.285.272.69", "id": 32, "temps": [15, 11, 14, 15, 9, 10, 12, 12, 12, 15, 17, 12], "co2_level": [1174, 1059, 1032, 1189, 1111, 1189]}}
3	dc-104	2019-07-18	▸ {"sensor-igauge": {"description": "Sensor attached to the container ceilings", "ip": "156.227.244.74", "id": 18, "temps": [7, 4, 4, 10, 9, 13, 3, 12, 13, 6, 8, 24], "co2_level": [981, 1173, 916, 1063, 1022, 1162]}, "sensor-ipad": {"description": "Sensor ipad attached to carbon cylinders", "ip": "83.186.150.137", "id": 26, "temps": [25, 19, 26, 17, 19, 22, 14, 19, 17, 18, 20, 14], "co2_level": [1009, 1407, 1346, 1266, 1078, 1146]}, "sensor-inest": {"description": "Sensor attached to the factory ceilings", "ip": "72.230.136.275", "id": 32, "temps": [20, 9, 15, 5, 10, 7, 20, 14, 7, 15, 7, 11], "co2_level": [1082, 1036, 798, 994, 1194, 1000]}, "sensor-istick": {"description": "Sensor embedded in exhaust pipes in the ceilings", "ip": "19.267.143.61", "id": 35, "temps": [19, 16, 19, 12, 19, 21, 19, 14, 21, 17, 24, 14], "co2_level": [1077, 962, 1093, 1279, 1035, 1044]}}

Showing all 3 rows.

Command took 7.45 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:15:49 PM on My Cluster

Explode a nested object

We can observe from the output that the `source` column contains a nested object with named `key-value` pairs. We'll use `EXPLODE` to get a closer look at the data in that column.

`EXPLODE` is used with arrays and elements of a map expression. When used with an array, it splits the elements into multiple rows. Used with a map, as in this example, it splits the elements of a map into multiple rows and columns and uses the default names, `key` and `value`, to name the new columns. This data structure is mapped such that each `key`, the name of a certain device, holds an object, `value`, containing information about that device.

Cmd 12

```
SELECT EXPLODE (source)
FROM DCDataRaw;
```

▸ (1) Spark Jobs

	key	value
1	sensor-igauge	▸ {"description": "Sensor attached to the container ceilings", "ip": "43.48.56.53", "id": 15, "temps": [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12], "co2_level": [1196, 1360, 1125, 1206, 1342, 1198]}
2	sensor-ipad	▸ {"description": "Sensor ipad attached to carbon cylinders", "ip": "171.59.65.289", "id": 21, "temps": [26, 17, 19, 13, 9, 12, 10, 12, 1, 13, 16, 12], "co2_level": [1002, 988, 1137, 1171, 1094, 1206]}
3	sensor-inest	▸ {"description": "Sensor attached to the factory ceilings", "ip": "189.125.254.115", "id": 29, "temps": [11, 13, 19, 8, 14, 16, 13, 14, 14, 9, 7, 12], "co2_level": [1266, 1254, 1169, 1204, 1165, 1342]}
4	sensor-istick	▸ {"description": "Sensor embedded in exhaust pipes in the ceilings", "ip": "27.84.20.121", "id": 37, "temps": [20, 18, 20, 18, 11, 14, 17, 24, 17, 15, 19, 22], "co2_level": [1098, 1113, 1161, 981, 967, 939]}
5	sensor-igauge	▸ {"description": "Sensor attached to the container ceilings", "ip": "286.139.183.97", "id": 18, "temps": [3, 1, 7, 8, 2, -4, 7, 1, 0, 8, -2, 3], "co2_level": [1494, 1385, 1378, 1335, 1480, 1107]}
6	sensor-ipad	▸ {"description": "Sensor ipad attached to carbon cylinders", "ip": "55.80.146.19", "id": 22, "temps": [11, 11, 16, 21, 8, 22, 13, 15, 10, 12, 14, 12], "co2_level": [1312, 1386, 1375, 1359, 1396, 1308]}
	sensor-inest	▸ {"description": "Sensor attached to the factory ceilings", "ip": "127.239.249.257", "id": 27, "temps": [21, 20, 23, 25, 22, 28, 23,

Truncated results, showing first 1000 rows.

Command took 1.56 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:18:06 PM on My Cluster

Common Table Expressions

Common Table Expressions (CTE) are supported in Spark SQL. A CTE provides a temporary result set which you can then use in a `SELECT` statement. These are different from temporary views in that they cannot be used beyond the scope of a single query. In this case, we will use the CTE to get a closer look at the nested data without writing a new table or view. CTEs use the `WITH` clause to start defining the expression.

Notice that after we explode the source column, we can access individual properties in the `value` field by using dot notation with the property name.

Cmd 14

```
WITH ExplodeSource  -- specify the name of the result set we will query
AS
(                   -- wrap a SELECT statement in parentheses
  SELECT            -- this is the temporary result set you will query
    dc_id,
    to_date(date) AS date,
    EXPLODE (source)
  FROM
    DCDataRaw
)
SELECT              -- write a select statment to query the result set
  key,
  dc_id,
  date,
  value.description,
  value.ip,
  value.temps,
  value.co2_level
FROM                -- this query is coming from the CTE we named
  ExplodeSource;

```

▸ (1) Spark Jobs

	key	dc_id	date	description	ip	temps	co2_level
1	sensor-igauge	dc-101	2019-07-01	Sensor attached to the container ceilings	43.48.56.53	▸ [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12]	▸ [1196, 1360, 1125, 1206, 1342, 1198]
2	sensor-ipad	dc-101	2019-07-01	Sensor ipad attached to carbon cylinders	171.59.65.289	▸ [26, 17, 19, 13, 9, 12, 10, 12, 1, 13, 16, 12]	▸ [1002, 988, 1137, 1171, 1094, 1206]
3	sensor-inest	dc-101	2019-07-01	Sensor attached to the factory ceilings	189.125.254.115	▸ [11, 13, 19, 8, 14, 16, 13, 14, 14, 9, 7, 12]	▸ [1266, 1254, 1169, 1204, 1165, 1342]

Create Table as Select (CTAS)

CTEs like those in the cell above are temporary and cannot be queried again. In the next cell, we demonstrate how you create a table using the common table expression syntax.

In Spark SQL, you can populate a new table with input data from a `SELECT` statement. The following is an example where we create a new table, `DeviceData`, using the CTE syntax we used in the previous cell. In this example, we rename the `key` column to `device_type`.

Cmd 16

```
DROP TABLE IF EXISTS DeviceData;
CREATE TABLE DeviceData
USING parquet
WITH ExplodeSource                      -- The start of the CTE from the last cell
AS
  (
  SELECT
  dc_id,
  to_date(date) AS date,
  EXPLODE (source)
  FROM DCDataRaw
  )
SELECT
  dc_id,
  key device_type,
  date,
  value.description,
  value.ip,
  value.temps,
  value.co2_level

FROM ExplodeSource;


```

▸ (1) Spark Jobs

OK

Command took 5.37 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:19:46 PM on My Cluster

Run a `SELECT` all to view the new table.

Cmd 18

```
SELECT * FROM DeviceData
```

▸ (1) Spark Jobs

	dc_id	device_type	date	description	ip	temps	co2_level
1	dc-101	sensor-igauge	2019-07-01	Sensor attached to the container ceilings	43.48.56.53	▸ [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12]	▸ [1196, 1360, 1125, 1206, 1342, 1198]
2	dc-101	sensor-ipad	2019-07-01	Sensor ipad attached to carbon cylinders	171.59.65.289	▸ [26, 17, 19, 13, 9, 12, 10, 12, 1, 13, 16, 12]	▸ [1002, 988, 1137, 1171, 1094, 1206]
3	dc-101	sensor-inest	2019-07-01	Sensor attached to the factory ceilings	189.125.254.115	▸ [11, 13, 19, 8, 14, 16, 13, 14, 14, 9, 7, 12]	▸ [1266, 1254, 1169, 1204, 1165, 1342]
4	dc-101	sensor-istick	2019-07-01	Sensor embedded in exhaust pipes in the ceilings	27.84.20.121	▸ [20, 18, 20, 18, 11, 14, 17, 24, 17, 15, 19, 22]	▸ [1098, 1113, 1161, 981, 967, 939]

Truncated results, showing first 1000 rows.

Command took 1.46 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:20:46 PM on My Cluster

Cmd 19

```
%run ../Includes/Classroom-Cleanup

```

Command took 0.05 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:20:49 PM on My Cluster

Cmd 20

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

Manipulating Data

In this notebook, you will be working with the online retail sales data that you worked with in the Module 3 Lab. This time, you will work with the columns of data that contain `NULL` values and a non-standard date format.

Run the following queries to learn about how to work with and manage null values and timestamps in Spark SQL. In this notebook, you will:

- Sample a table
- Access individual values from an array
- Reformat values using a padding function
- Concatenate values to match a standard format
- Access parts of a `DateType` value like the month, day, or year

Cmd 3

Getting Started

Run the cell below to set up your classroom environment.

Cmd 4

```
%run ../Includes/Classroom-Setup
```

Command took 6.38 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:24:46 PM on My Cluster

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false

res2: Boolean = false
```

Create table

Our data is stored as a csv. The optional arguments show the path to the data and store the first row as a header.

Cmd 6

```
DROP TABLE IF EXISTS outdoorProductsRaw;
CREATE TABLE outdoorProductsRaw USING csv OPTIONS (
  path "/mnt/training/online_retail/data-001/data.csv",
  header "true"
)
```

▸ (1) Spark Jobs

OK

Command took 1.91 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:26:55 PM on My Cluster

Cmd 7

Describe

Recall that the `DESCRIBE` command tells us about the schema of the table. Notice that all of our columns are string values.

Cmd 8

```
DESCRIBE outdoorProductsRaw
```

	col_name	data_type	comment
1	InvoiceNo	string	null
2	StockCode	string	null
3	Description	string	null
4	Quantity	string	null
5	InvoiceDate	string	null
6	UnitPrice	string	null
7	CustomerID	string	null

Showing all 8 rows.

Command took 0.14 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:26:58 PM on My Cluster

Sample the table

In the previous reading, you accessed a random sample of rows from a table using the `RAND()` function and `LIMIT` keyword. While this is a common way to retrieve a sample with other SQL dialects, Spark SQL includes a built-in function that you may want to use instead.

The function, `TABLESAMPLE` , allows you to return a number of rows or a certain percentage of the data. In the cell directly below this one, we show that `TABLESAMPLE` can be used to access a specific number of rows. In the following cell, we show that it can be used to access a given percentage of the data. Please note, however, any table display is limited to 1,000 rows. If the percentage of data you request returns more thna 1,000 rows, only the first 1000 will show.

The sample that displays 2 percent of the table is also ordered by the `InvoiceDate` . This shows off a formatting issue in the date column that we will have to fix later on. Take a moment and see if you can predict how we might need to change in the way the `InvoiceDate` is written.

Cmd 10

```
SELECT * FROM outdoorProductsRaw TABLESAMPLE (5 ROWS)
```

▸ (1) Spark Jobs

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850	United Kingdom
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850	United Kingdom

Showing all 5 rows.

Command took 1.00 second -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:27:51 PM on My Cluster

```
1  SELECT * FROM outdoorProductsRaw TABLESAMPLE (2 PERCENT) ORDER BY InvoiceDate
```

▸ (1) Spark Jobs

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	540561	82494L	WOODEN FRAME ANTIQUE WHITE	6	1/10/11 10:32	2.95	13004	United Kingdom
2	540566	22467	GUMBALL COAT RACK	2	1/10/11 10:58	2.55	17811	United Kingdom
3	540566	22521	CHILDS GARDEN TROWEL PINK	2	1/10/11 10:58	0.85	17811	United Kingdom
4	540567	82600	NO SINGING METAL SIGN	12	1/10/11 11:09	2.1	16850	United Kingdom
5	540646	22730	ALARM CLOCK BAKELIKE IVORY	1	1/10/11 14:32	7.62	null	United Kingdom
6	540646	21700	BIG DOUGHNUT FRIDGE MAGNETS	1	1/10/11 14:32	8.47	null	United Kingdom
7	540646	22988	SOLDIERS EGG CUP	2	1/10/11 14:32	2.51	null	United Kingdom

Truncated results, showing first 1000 rows.

Command took 1.81 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:27:54 PM on My Cluster

Cmd 12

Check for null values

Run this cell to see the number of `NULL` values in the `Description` column of our table.

Cmd 13

```
1  SELECT count(*) FROM outdoorProductsRaw WHERE Description IS NULL;
2
```

▸ (2) Spark Jobs

	count(1)
1	166

Showing all 1 rows.

Command took 2.58 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:27:56 PM on My Cluster

Create a temporary view

The next cell creates the temporary view `outdoorProducts`. By now, you should be familiar with how to create (or replace) a temporary view. There are a few new commands to notice in this particular command.

This is where we will start to work with the problematic date formatting mentioned previously. Did you notice the inconsistency in your displays?

Our dates do not have a standard number of digits for months and years. For example, `12/1/11` has a two-digit month and one-digit day, while `1/10/11` has an one-digit month and two-digit day. It's easy enough to specify a format to convert a string to a date, but the format must be consistent throughout the table. We will begin to attempt a fix for this problem by simply separating all of the components of the date and dropping the time value entirely.

Cmd 15

Code breakdown

`COALESCE` - This command is popular among many different SQL dialects. We can use it to replace `NULL` values. For all `NULL` values in the `Description` column, `COALESCE()` will replace the null with a value you include in the function. In this case, the value is `"Misc"`. For more information about `COALESCE`, check the documentation.

`SPLIT` - This command splits a string value around a specified character and returns an **array**. An array is a list of values that you can access by position. In this case, the forward slash ("/") is the character we use to split the data. The first value in the array is the month. This list is **zero-indexed** for the index of the first position is **0**. Since we want to pull out the first value as the month, we indicate the value like this: `SPLIT(InvoiceDate, "/")[0]` and rename the column `month`. The day is the second value and its index is 1.

The third `SPLIT` is different. Remember that our `InvoiceDate` column is a string that includes a date and time. Each part of the date is seperated by a forward slash, but between the date and the time, there is only a space. `Line 10` contains a **nested** `SPLIT` function that splits the string on a space delimiter.

`SPLIT(InvoiceDate, " ")[0]` --> Drops the time from the string and leaves the date intact. Then, we split that value on the forward slash delimiter. We access the year at index 2. Learn more about the `SPLIT` function by accessing the documentation.

```
CREATE
OR REPLACE TEMPORARY VIEW outdoorProducts AS
SELECT
  InvoiceNo,
  StockCode,
  COALESCE(Description, "Misc") AS Description,
  Quantity,
  SPLIT(InvoiceDate, "/")[0] month,
  SPLIT(InvoiceDate, "/")[1] day,
  SPLIT(SPLIT(InvoiceDate, " ")[0], "/")[2] year,
  UnitPrice,
  Country
FROM
  outdoorProductsRaw
```

OK

Command took 0.21 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:29:26 PM on My Cluster

Cmd 17

Check "Misc" values

We perform a quick sanity check here to demonstrate that all of the `NULL` values in Description have been replaced with the string `"Misc"`.

Cmd 18

```
SELECT count(*) FROM outdoorProducts WHERE Description = "Misc"
```

▸ (2) Spark Jobs

	count(1)
1	166

Showing all 1 rows.

Command took 1.53 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:30:02 PM on My Cluster

Create a new table

Now, we can write a new table with a consistently formatted date string. Notice that this table creation statement has a CTE inside of it. Recall that the CTE starts with a `WITH` clause.

Notice the `LPAD()` functions on lines 11 and 12. This function inserts characters to the left of a string until the string reachers a certain length. In this example, we use `LPAD` to insert a zero to the left of any value in the month or day column that **is not** two digits. For values that are two digits, `LPAD` does nothing.

We use the `padStrings` CTE to standardize the length of the individual date components. When we query the CTE, we use the `CONCAT_WS()` function to put the date string back together. This function returns a concatenated string with a specified separator. In this example, we concatenate values from the month, date, and year columns, and specify that each value should be separated by a forward slash ("/").

Cmd 20

```
DROP TABLE IF EXISTS standardDate;
CREATE TABLE standardDate

WITH padStrings AS
(
SELECT
  InvoiceNo,
  StockCode,
  Description,
  Quantity,
  LPAD(month, 2, 0) AS month,
  LPAD(day, 2, 0) AS day,
  year,
  UnitPrice,
  Country
FROM outdoorProducts
)
SELECT
 InvoiceNo,
  StockCode,
  Description,
  Quantity,
  concat_ws("/", month, day, year) sDate,
  UnitPrice,
  Country
FROM padStrings;
```

Table check

When we view our new table, we can see that the date field shows two digits each for the month, day, and year.

Cmd 22

```
1  SELECT * FROM standardDate LIMIT 5;
```

▸ (1) Spark Jobs

	InvoiceNo	StockCode	Description	Quantity	sDate	UnitPrice	Country
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/01/10	2.55	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	12/01/10	3.39	United Kingdom
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/01/10	2.75	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/01/10	3.39	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/01/10	3.39	United Kingdom

Showing all 5 rows.

Command took 1.40 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:32:27 PM on My Cluster

Cmd 23

Check schema

Oops! All of our values are still strings. The date field would be much more useful as a `DateType`.

Cmd 24

```
1  DESCRIBE standardDate;
```

	col_name	data_type	comment
1	InvoiceNo	string	
2	StockCode	string	
3	Description	string	
4	Quantity	string	
5	sDate	string	

Change to DateType

In the next cell, we create a new temporary view that converts the value to a date. The optional argument `MM/dd/yy` indicates the meaning of each part of the date. You can find a complete guide to Spark SQL's Datetime Patterns here.

Also, we cast the `UnitPrice` as a `DOUBLE` so that we can treat it as a number.

Cmd 26

```
1  CREATE
2  OR REPLACE TEMPORARY VIEW salesDateFormatted AS
3  SELECT
4    InvoiceNo,
5    StockCode,
6    to_date(sDate, "MM/dd/yy") date,
7    Quantity,
8    CAST(UnitPrice AS DOUBLE)
9  FROM
10   standardDate
```

OK

Command took 0.18 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:32:33 PM on My Cluster

Visualize Data

We can extract the day of the week and figure out the total quantitybar of items sold on each day. You can create a quick visual by clicking on the chart icon and creating a bar chart where the key is the `day` and the values are the `quantity`.

We use the `date_format()` function to map the day to a day of the week. This function converts a `timestamp` to a `string` in the format specified. For this command, the `"E"` specifies that we want the output to be the day of the week.

Cmd 28

```
SELECT
  date_format(date, "E") day,
  SUM(quantity) totalQuantity
FROM
  salesDateFormatted
GROUP BY (day)
ORDER BY day
```

▸ (2) Spark Jobs

	day	totalQuantity
1	Fri	90762
2	Mon	76366
3	Sun	43117
4	Thu	114827
5	Tue	106256
6	Wed	116652

Showing all 6 rows.

Command took 3.24 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:33:20 PM on My Cluster

Cmd 29

```
%run ../Includes/Classroom-Cleanup
```

Command took 0.04 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:34:02 PM on My Cluster

Visualize Data

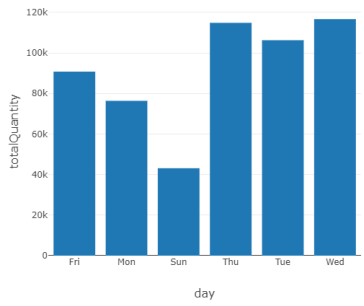
We can extract the day of the week and figure out the total quantitybar of items sold on each day. You can create a quick visual by clicking on the chart icon and creating a bar chart where the key is the `day` and the values are the `quantity`.

We use the `date_format()` function to map the day to a day of the week. This function converts a `timestamp` to a `string` in the format specified. For this command, the `"E"` specifies that we want the output to be the day of the week.

Cmd 28

```
SELECT
  date_format(date, "E") day,
  SUM(quantity) totalQuantity
FROM
  salesDateFormatted
GROUP BY (day)
ORDER BY day
```

▸ (2) Spark Jobs



Lab 2 - Data Munging

Module 5 Assignment

In this exercise, you will be working with mock data meant to replicate data from an ecommerce mattress seller. Broadly, your work is to clean up and present this data so that it can be used to target geographic areas. Work through the tasks below and answer the challenge to produce the required report.

☆ In this assignment you will:

- Work with hierarchical data
- Use common table expressions to display data
- Create new tables based on existing tables
- Manage working with null values and timestamps

As you work through the following tasks, you will be prompted to enter selected answers in Coursera. Find the quiz associated with this lab to enter your answers.

Run the cell below to prepare this workspace for the lab.

Cmd 3

```
%run ../Includes/Classroom-Setup
```

Command took 5.97 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:48:56 PM on My Cluster

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false

res2: Boolean = false
```

Exercise 1: Create a table

Summary: Create a new table named `eventsRaw`

Use this path to access the data: `/mnt/training/ecommerce/events/events.parquet`

Steps to complete:
- Make sure this notebook is idempotent by first dropping the table named `eventsRaw`, if it exists already
- Use the provided path to read in the data

Cmd 5

```
--TODO
drop table if exists eventsRaw;
create table eventsRaw
using parquet
options (
  path "/mnt/training/ecommerce/events/events.parquet"
)
```

▸ (1) Spark Jobs

OK

```
Command took 1.02 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:51:39 PM on My Cluster
```

Cmd 6

Exercise 2: Understand the schema and metadata

Summary: Run a command to display this table's schema and other detailed table information

Notice that this table includes `ArrayType` and `StructType` data

Steps to complete:
- Run a single command to display the table information
- **Answer the corresponding question in Coursera, in the quiz for this module, regarding the location of this table**

```
describe extended eventsRaw;
```

	col_name	data_type	comment
18	Created By	Spark 3.1.0	
19	Type	EXTERNAL	
20	Provider	parquet	
21	Location	dbfs:/mnt/training/ecommerce/events/events.parquet	
22	Serde Library	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe	
23	InputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat	
24	OutputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat	

Showing all 24 rows.

```
Command took 0.17 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 9:53:16 PM on My Cluster
```

Cmd 8

Exercise 3: Sample the table

Summary: Sample this table to get a closer look at the data

Steps to complete:
- Sample the table to display up to 1 percent of the records

Cmd 9

```
--TODO
Select * from eventsRaw tablesample (1 percent)
```

▸ (1) Spark Jobs

	device	ecommerce	event_name	event_previous_timestamp	event_timestamp	geo	items
1	Windows	▸ {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	mattresses	null	1593878628143633	▸ {"city": "Cottage Grove", "state": "MN"}	[]
2	iOS	▸ {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	original	1593878068949001	1593878170903989	▸ {"city": "Longview", "state": "WA"}	[]
3	iOS	▸ {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	main	null	1593876967068480	▸ {"city": "Fresno", "state": "CA"}	[]
4	Android	▸ {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	main	null	1593877461533427	▸ {"city": "Frisco", "state": "TX"}	[]
5	Android	▸ {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	pillows	null	1593877831444924	▸ {"city": "Jurupa Valley", "state": "CA"}	[]
6	macOS	▸ {"purchase_revenue_in_usd": null, "total_item_quantity": null, "unique_items": null}	checkout	1593877098197691	1593877457933580	▸ {"city": "Mounds View", "state": "MN"}	▸ [{"coupon": null, "item_id": "M_STAN_K", "item_name": "Sta

Exercise 4: Create a new table

Summary: Create a table `purchaseEvents` that includes event data *with* purchases that has the following schema:

ColumnName	DataType
purchases	double
previousEventDate	date
eventDate	date
city	string
state	string
userId	string

⚠ The timestamps in this table are meant to match those used in Google Analytics, which measures time to the microsecond. To convert to unixtime, you must divide these values by 1000000 (10e6) before casting to a timestamp.

Hint: Access values from StructType objects using dot notation

Steps to complete:

- Make sure this notebook is idempotent by first dropping the table, if it exists
- Create a table based on the existing table
- Use a common table expression to manipulate your data before writing the `SELECT` statement that will define your table *(Recommended)*
- Do not include records where the `purchase_revenue_in_usd` is `NULL`
- Sort the table so that the city and state with the greatest total purchase is listed first

```
1  --TODO
2  drop table if exists purchaseEvents;
3  Create table purchaseEvents
4  Using parquet
5  as
6  ( Select user_id as userId,
7           cast(event_previous_timestamp/1000000 as timestamp) as previousEventDate,
8           cast(event_timestamp/1000000 as timestamp) as eventDate,
9           ecommerce.purchase_revenue_in_usd as purchases,
10          geo.city as city,
11          geo.state as state
12    from
13      eventsRaw
14 Where
15   ecommerce.purchase_revenue_in_usd is not null
16 Order by purchases desc
17 );
18
```

▸ (3) Spark Jobs

OK

1

Command took 23.64 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 11:13:47 PM on My Cluster

Exercise 5: Count the records

Summary: Count all the records in your new table.

Steps to complete:

- Write a `SELECT` statement that counts the records in `purchaseEvents`
- **Answer the corresponding quiz question in Coursera**

Cmd 13

```
1  --TODO
2  select count(*)
3  from purchaseEvents;
```

▸ (2) Spark Jobs

	count(1)
1	180678

Showing all 1 rows.

Command took 1.50 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 11:12:56 PM on My Cluster

Exercise 6: Find the location with the top purchase

Summary: Write a query to produce the city and state where the top purchase amount originated.

Steps to complete:

- Write a query, sorted by `purchases`, that shows the city and state of the top purchase
- **Answer the corresponding quiz question in Coursera**

Cmd 15

```
1  --TODO
2  select *
3  from purchaseEvents
4  order by purchases desc
5  limit 5;
```

▸ (1) Spark Jobs

	userId	previousEventDate	eventDate	purchases	city	state
1	UA000000103389917	2020-06-19T15:57:03.664+0000	2020-06-19T15:57:06.664+0000	5830	Amarillo	TX
2	UA000000104420407	2020-06-22T21:51:00.164+0000	2020-06-22T21:51:09.375+0000	5485	Tampa	FL
3	UA000000106350334	2020-06-30T18:54:36.577+0000	2020-06-30T18:55:08.044+0000	5289	Buffalo	NY
4	UA000000105459671	2020-06-30T13:18:19.009+0000	2020-06-30T13:28:02.510+0000	5219.1	Miami	FL
5	UA000000106281438	2020-06-30T16:27:33.169+0000	2020-06-30T16:45:39.976+0000	5180	Sarasota	FL

Showing all 5 rows.

Command took 2.28 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 11:14:49 PM on My Cluster

Challenge: Produce reports

Summary: Use the `purchaseEvents` table to produce queries that explore purchase patterns in the table. Add visualizations to a dashboard to produce one comprehensive customer report.

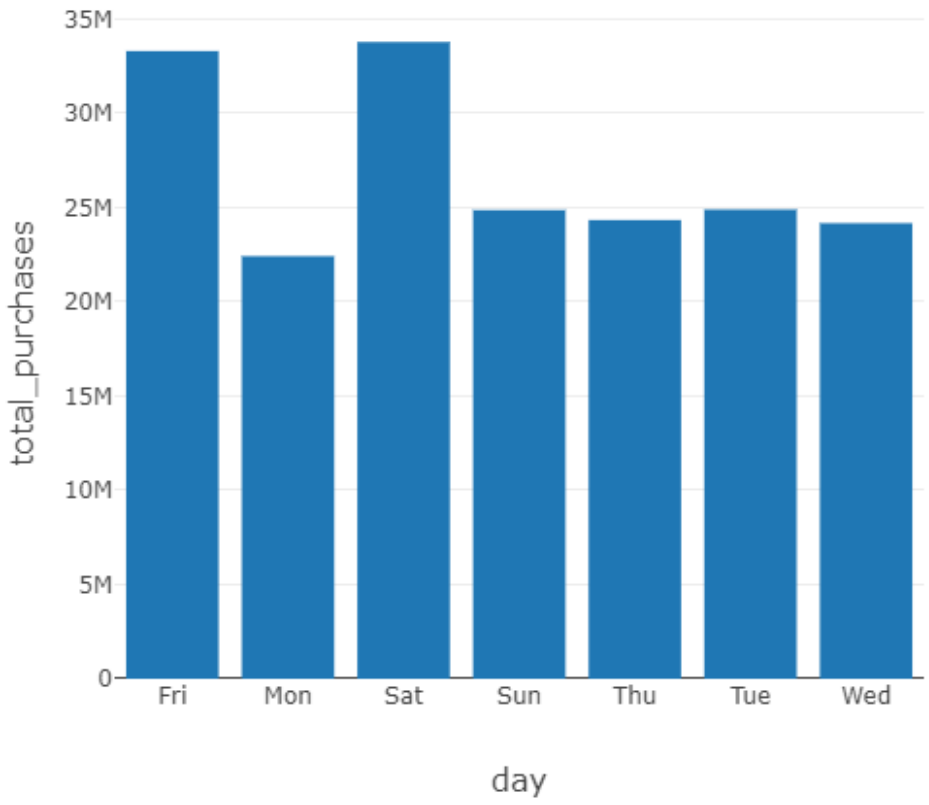
Steps to complete:

- Create visualizations to report on:
  - total purchases by day of week
  - average purchases by date of purchase
  - total purchases by state
  - Any other patterns you can find in the data
- Join your table with the data at the path listed below to get list of customers with confirmed email addresses
- **Answer the corresponding quiz question in Coursera**

Access the data that holds user email addresses. You can read the data from this path: `/mnt/training/ecommerce/users/users.parquet`

```sql
-- Total purcases by day of week
select date_format(eventDate, "E") day, sum(purchases) as total_purchases
from purchaseEvents
group by (day)
order by day;
```

▸ (2) Spark Jobs

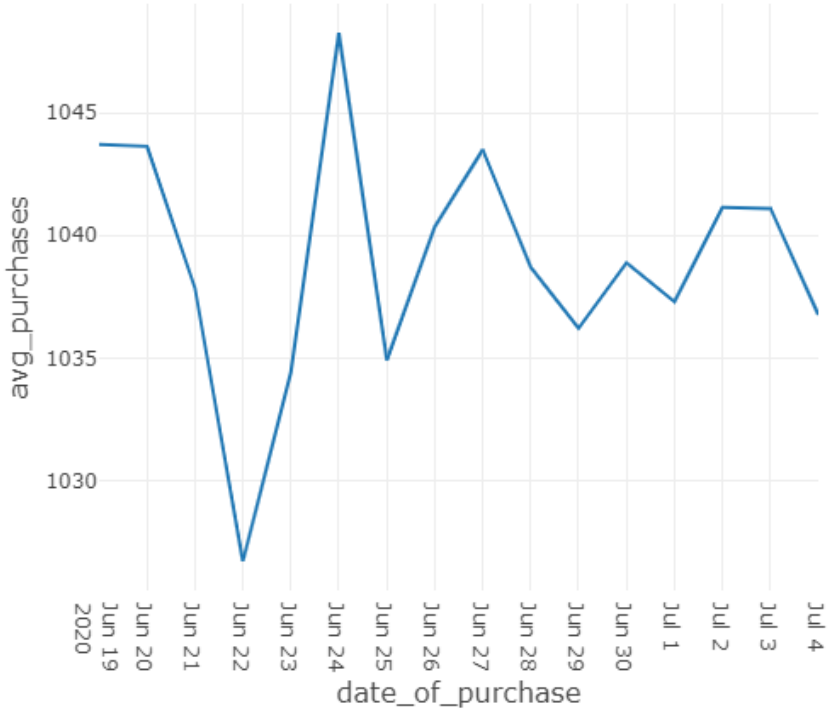


Plot Options...

Command took 2.14 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 11:32:10 PM on My Cluster

```sql
-- Average of purchases by date of purchase
select to_date(eventDate, "MM/dd/yy") date_of_purchase, avg(purchases) as avg_purchases
from purchaseEvents
group by date_of_purchase
order by date_of_purchase;
```

▸ (2) Spark Jobs

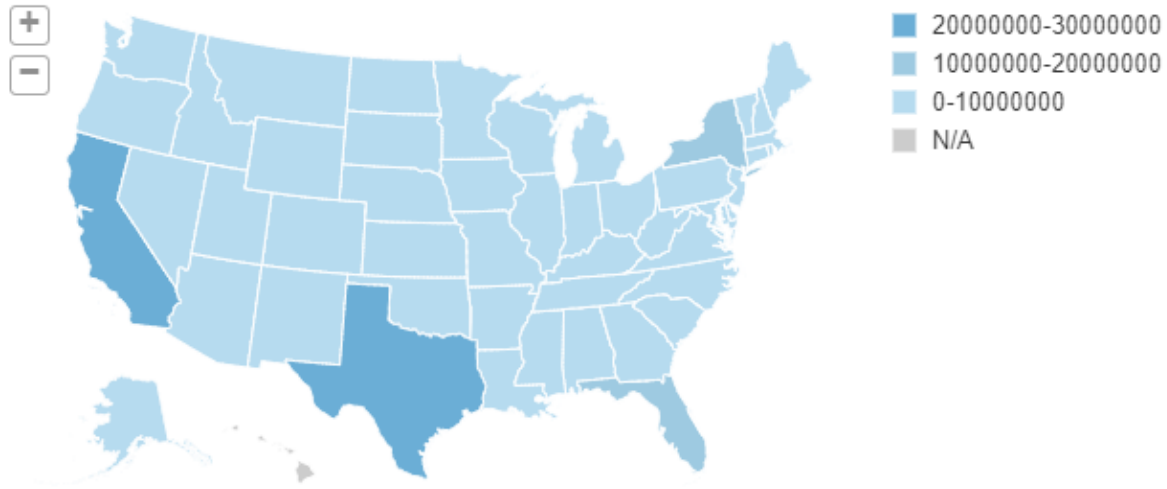


Plot Options...

Command took 2.18 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 11:28:16 PM on My Cluster

```sql
-- Total purchases by state
select state, sum(purchases) as total_purchases
from purchaseEvents
group by state
order by total_purchases desc
```

▸ (2) Spark Jobs



Plot Options...

Command took 2.12 seconds -- by SWCPROPERTY@GMAIL.COM at 7/3/2021, 11:24:31 PM on My Cluster

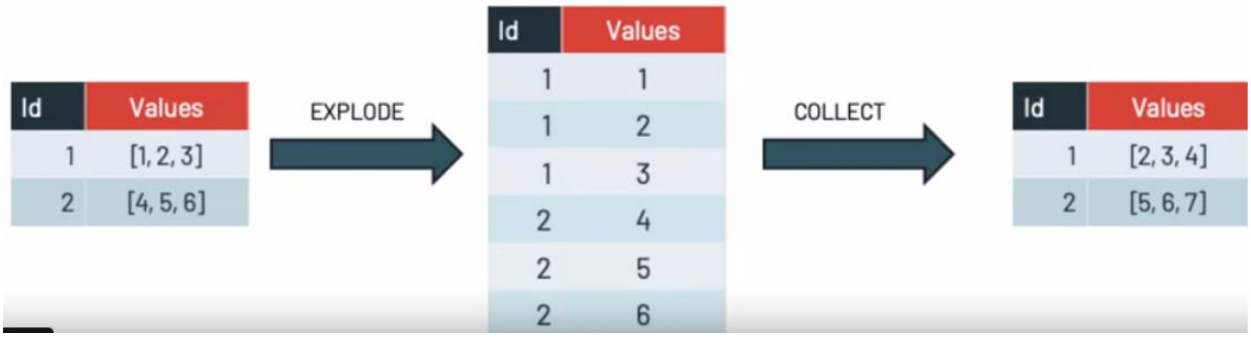
Complex data types: Arrays

- Sequence of values
- Each value is an **element** of an array
- Can contain nulls

id	values
1	[1, 2, 3]
2	[4, 5, 6]

Manipulating complex data types: Past approaches

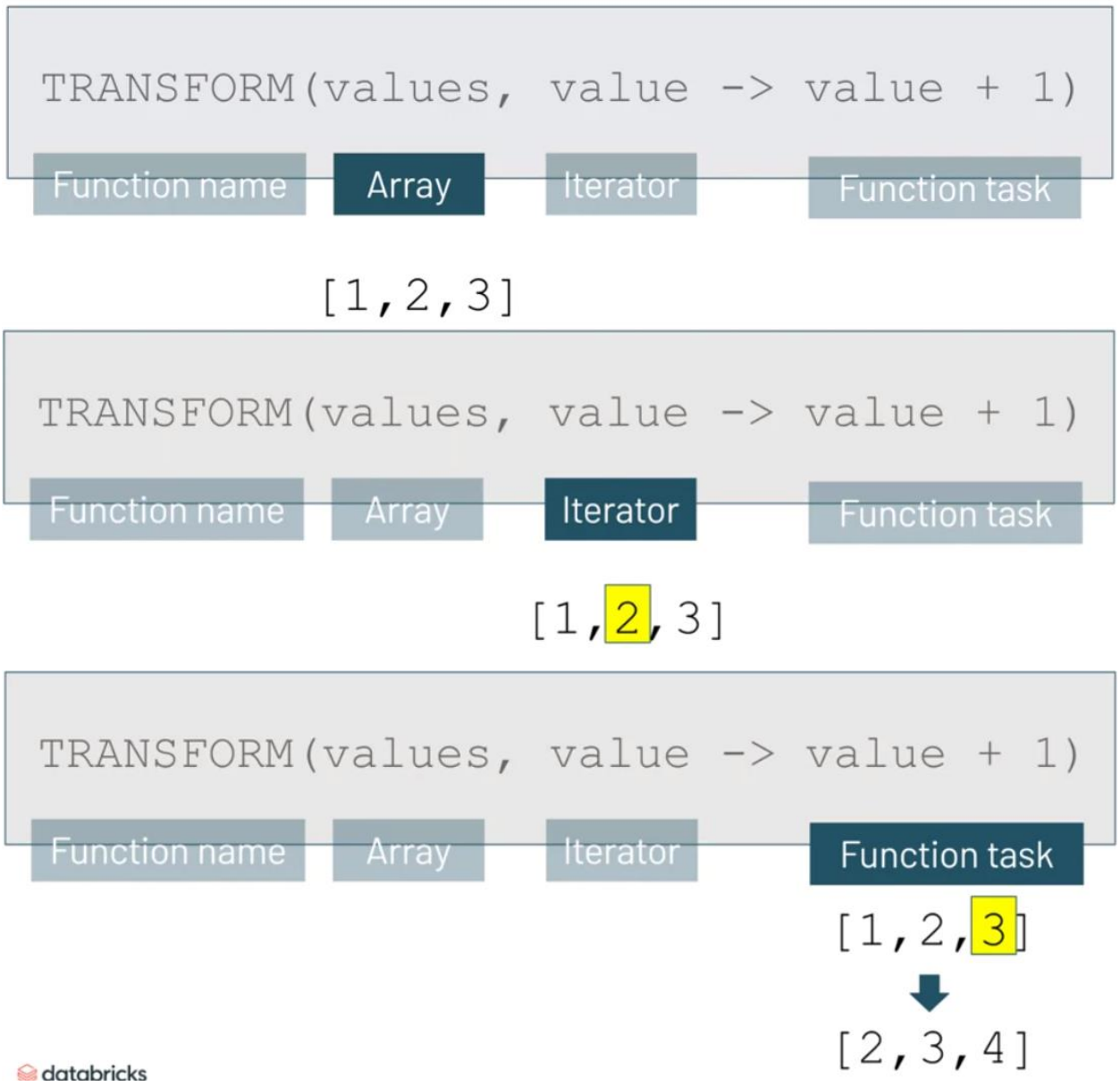
- Explode the nested structure into individual rows, apply functions, re-create structure



Explode and Collect

- Highly error prone
- Repacking arrays in order is not guaranteed
- Inefficient

Higher-order Functions



6.1 Higher-order Functions (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revis

Cmd 2

Higher-order functions

Higher order functions in Spark SQL allow you to work directly with complex data types. When working with hierarchical data, as we were in the previous lesson and lab, records are frequently stored as array or map type objects. This lesson will demonstrate how to use higher-order functions to transform, filter, and flag array data while preserving the original structure. In this notebook, we work strictly with arrays of strings; in the subsquent notebook, you will work with more functions and numerical data. Skilled application of these functions can make your work with this kind of data faster, more powerful, and more reliable.

In this notebook, you will:

- Apply higher-order functions (`TRANSFORM` , `FILTER` , `EXISTS`) to arrays of strings

Run the following queries to learn about how to work with higher-order functions.

Cmd 3

Getting Started

Run the cell below to set up your classroom environment

Cmd 4

```
%run ../Includes/Classroom-Setup
```

```
Command took 1.37 minutes -- by SWCPROPERTY@GMAIL.COM at 7/4/2021, 9:39:17 PM on My Cluster
```

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false

res2: Boolean = false
```

Working with Text Data

We can use higher-order functions to easily work with arrays of text data. The exercises in this section are meant to demonstrate the `TRANSFORM` , `FILTER` , and `EXISTS` functions to manipulate data and create flags for when a value does or does not exist.

These examples use data collected about Databricks blog posts. Run the cell below to create the table. Then, run the next cell to view the schema.

In this data set, the `authors` and `categories` columns are both ArrayType; we'll be using these columns with higher-order functions.

Cmd 6

```
CREATE TABLE IF NOT EXISTS DatabricksBlog
  USING json
  OPTIONS (
    path "dbfs:/mnt/training/databricks-blog.json",
    inferSchema "true"
  )
```

▸ (1) Spark Jobs

OK

```
Command took 5.72 seconds -- by SWCPROPERTY@GMAIL.COM at 7/4/2021, 9:51:27 PM on My Cluster
```

Cmd 7

```
DESCRIBE DatabricksBlog
```

	col_name	data_type	comment
1	authors	array<string>	null
2	categories	array<string>	null
3	content	string	null
4	creator	string	null
5	dates	struct<createdOn:string,publishedOn:string,tz:string>	null
6	description	string	null
7	id	bigint	null

Showing all 11 rows.

```
Command took 1.01 seconds -- by SWCPROPERTY@GMAIL.COM at 7/4/2021, 9:51:32 PM on My Cluster
```

Filter

Filter allows us to create a new column based on whether or not values in an array meet a certain condition. Let's say we want to remove the category `"Engineering Blog"` from all records in our `categories` column. I can use the `FILTER` function to create a new column that excludes that value from the each array.

Let's dissect this line of code to better understand the function:

```
FILTER (categories, category -> category <> "Engineering Blog") woEngineering
```

`FILTER` : the name of the higher-order function
`categories` : the name of our input array
`category` : the name of the iterator variable. You choose this name and then use it in the lambda function. It iterates over the array, cycling each value into the function one at a time.
`->` : Indicates the start of a function
`category <> "Engineering Blog"` : This is the function. Each value is checked to see if it **is different** than the value `"Engineering Blog"` . If it is, it gets filtered into the new column, `woEngineering` .

Cmd 9

```
SELECT
  categories,
  FILTER (categories, category -> category <> "Engineering Blog") woEngineering
FROM DatabricksBlog

```

▸ (1) Spark Jobs

	categories	woEngineering
1	▸ ["Company Blog", "Partners"]	▸ ["Company Blog", "Partners"]
2	▸ ["Apache Spark", "Engineering Blog", "Machine Learning"]	▸ ["Apache Spark", "Machine Learning"]
3	▸ ["Company Blog", "Partners"]	▸ ["Company Blog", "Partners"]
4	▸ ["Apache Spark", "Engineering Blog"]	▸ ["Apache Spark"]
5	▸ ["Apache Spark", "Engineering Blog"]	▸ ["Apache Spark"]
6	▸ ["Apache Spark", "Ecosystem", "Engineering Blog"]	▸ ["Apache Spark", "Ecosystem"]
7	▸ ["Company Blog", "Customers"]	▸ ["Company Blog", "Customers"]

Showing all 365 rows.

```
Command took 2.43 seconds -- by SWCPROPERTY@GMAIL.COM at 7/4/2021, 9:52:16 PM on My Cluster
```

Filter, subqueries, and `WHERE`

You may write a filter that produces a lot of empty arrays in the created column. When that happens, it can be useful to use a `WHERE` clause to show only non-empty array values in the returned column.

In this example, we accomplish that by using a subquery. A **subquery** in SQL is a query within a query. They are useful for performing an operations in multiple steps. In this case, we're using it to create the named column that we will use with a `WHERE` clause.

Cmd 11

```
SELECT
  *
FROM
  (
    SELECT
      authors, title,
      FILTER(categories, category -> category = "Engineering Blog") AS blogType
    FROM
      DatabricksBlog
  )
WHERE
  size(blogType) > 0
```

▸ (1) Spark Jobs

	authors	title	blogType
1	▸ ["Tathagata Das"]	Apache Spark 0.9.1 Released	▸ ["Engineering Blog"]
2	▸ ["Michael Armbrust", "Reynold Xin"]	Spark SQL: Manipulating Structured Data Using Apache Spark	▸ ["Engineering Blog"]
3	▸ ["Patrick Wendell"]	Apache Spark 0.9.0 Released	▸ ["Engineering Blog"]
4	▸ ["Ali Ghodsi", "Ahir Reddy"]	Apache Spark In MapReduce (SIMR)	▸ ["Engineering Blog"]

Showing all 141 rows.

Command took 1.28 seconds -- by SWCPROPERTY@GMAIL.COM at 7/4/2021, 9:53:00 PM on My Cluster

Exists

Exists tests whether a statement is true for one or more elements in an array. Let's say we want to flag all blog posts with `"Company Blog"` in the categories field. I can use the `EXISTS` function to mark which entries include that category.

Let's dissect this line of code to better understand the function:

`EXISTS (categories, c -> c = "Company Blog") companyFlag`

`EXISTS` : the name of the higher-order function
`categories` : the name of our input array
`c` : the name of the iterator variable. You choose this name and then use it in the lambda function. It iterates over the array, cycling each value into the function one at a time. Note that we're using the same kind as references as in the previous command, but we name the iterator with a single letter
`->` : Indicates the start of a function
`c = "Engineering Blog"` : This is the function. Each value is checked to see if it **is the same as** the value `"Company Blog"`. If it is, it gets flagged into the new column, `companyFlag`

Cmd 13

```
SELECT
  categories,
  EXISTS (categories, c -> c = "Company Blog") companyFlag
FROM DatabricksBlog

```

▸ (1) Spark Jobs

	categories	companyFlag
1	▸ ["Company Blog", "Partners"]	true
2	▸ ["Apache Spark", "Engineering Blog", "Machine Learning"]	false
3	▸ ["Company Blog", "Partners"]	true
4	▸ ["Apache Spark", "Engineering Blog"]	false
5	▸ ["Apache Spark", "Engineering Blog"]	false
6	▸ ["Apache Spark", "Ecosystem", "Engineering Blog"]	false
7	▸ ["Company Blog", "Customers"]	true

Showing all 365 rows.

Command took 0.84 seconds -- by SWCPROPERTY@GMAIL.COM at 7/4/2021, 9:53:40 PM on My Cluster

Transform

Transform uses the provided function to transform all elements of an array. SQL's built-in functions are designed to operate on a single, simple data type within a cell. They cannot process array values. Transform can be particularly useful when you want to apply an existing function to each element in an array. In this case, we want to rewrite all of the names in the `categories` column in lowercase.

Let's dissect this line of code to better understand the function:

`TRANSFORM(categories, cat -> LOWER(cat)) lwrCategories`

`TRANSFORM` : the name of the higher-order function
`categories` : the name of our input array
`cat` : the name of the iterator variable. You choose this name and then use it in the lambda function. It iterates over the array, cycling each value into the function one at a time. Note that we're using the same kind as references as in the previous command, but we name the iterator with a new variable
`->` : Indicates the start of a function
`LOWER(cat)` : This is the function. For each value in the input array, the built-in function `LOWER()` is applied to transform the word to lowercase.

Cmd 15

```
SELECT
  TRANSFORM(categories, cat -> LOWER(cat)) lwrCategories
FROM DatabricksBlog
```

▸ (1) Spark Jobs

	lwrCategories
1	["company blog", "partners"]
2	["apache spark", "engineering blog", "machine learning"]
3	["company blog", "partners"]
4	["apache spark", "engineering blog"]
5	["apache spark", "engineering blog"]
6	["apache spark", "ecosystem", "engineering blog"]
7	["company blog", "customers"]

Showing all 365 rows.

Command took 0.86 seconds -- by SWCPROPERTY@GMAIL.COM at 7/4/2021, 9:54:18 PM on My Cluster

6.2 Aggregating and Summarizing Data (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revision hi

Aggregating and summarizing data

Now let's look at some powerful functions we can use to aggregate and summarize data. In this notebook, we will continue to work with hihger-order functions; this time we will apply them to arrays containing numerical data. Also, we will work with additional functions in Spark SQL that can be helpful when presenting data.

In this notebook, you will:

- Apply higher-order functions to numeric data
- Use a `PIVOT` command to create Pivot tables
- Use `ROLLUP` and `CUBE` modifiers to generate subtotals
- Use window functions to perform operations on a group of rows
- Use Databricks visualization tools to visualize and share data

Run the cell below to set up our classroom environment.

Cmd 3

```
%run ../Includes/Classroom-Setup
```

Command took 5.86 minutes -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:19:37 AM on My Cluster

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false

res2: Boolean = false
```

Cmd 4

Higher-order functions and numerical data

Each of the higher-order functions we worked with in the last lesson can also be used with numerical data. In this lesson, we demonstrate how each of the functions in the previous lesson work with numeric data, as well as explore some powerful new higher-order functions.

Run the next two cells to create and describe the table we will be working with. You may recognize this table from a previous lesson. Recall that it contains data measuring environmental variability in a collection of data centers. The table `DeviceData` contains the `temps` and `co2Level` arrays we use to demonstrate higher-order functions.

```sql
DROP TABLE IF EXISTS DCDataRaw;
CREATE TABLE DCDataRaw
USING parquet
OPTIONS (
    PATH "/mnt/training/iot-devices/data-centers/2019-q2-q3"
    );

CREATE TABLE IF NOT EXISTS DeviceData2
USING parquet
WITH ExplodeSource
AS
  (
  SELECT
  dc_id,
  to_date(date) AS date,
  EXPLODE (source)
  FROM DCDataRaw
  )
SELECT
  dc_id,
  key device_type,
  date,
  value.description,
  value.ip,
  value.temps,
  value.co2_level co2Level

FROM ExplodeSource;
```

▸ (2) Spark Jobs

OK

Command took 13.13 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:34:02 AM on My Cluster

```
1  DESCRIBE DeviceData2;
```

	col_name	data_type	comment
1	dc_id	string	null
2	device_type	string	null
3	date	date	null
4	description	string	null
5	ip	string	null
6	temps	array<int>	null
7	co2Level	array<int>	null

Showing all 7 rows.

Command took 0.18 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:34:25 AM on My Cluster

Cmd 7

Preview data

Let's take a look a sample fo the data so that we con better understand the array values.

Cmd 8

```
1  SELECT
2    temps,
3    co2Level
4  FROM DeviceData2
5  TABLESAMPLE (1 ROWS)
```

▸ (1) Spark Jobs

	temps	co2Level
1	▸ [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12]	▸ [1196, 1360, 1125, 1206, 1342, 1198]

Showing all 1 rows.

Command took 1.65 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:35:23 AM on My Cluster

Filter

Filter operates on arrays containing numeric data just the same as those with text data. In this case, let's imagine that we want to collect all temperatures above a given threshold. Run the cell below to view the example.

Cmd 10

```
SELECT
  temps,
  FILTER(temps, t -> t > 18) highTemps
FROM DeviceData2
```

▸ (1) Spark Jobs

	temps	highTemps
1	▸ [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12]	▸ [19, 23]
2	▸ [26, 17, 19, 13, 9, 12, 10, 12, 1, 13, 16, 12]	▸ [26, 19]
3	▸ [11, 13, 19, 8, 14, 16, 13, 14, 14, 9, 7, 12]	▸ [19]
4	▸ [20, 18, 20, 18, 11, 14, 17, 24, 17, 15, 19, 22]	▸ [20, 20, 24, 19, 22]

Truncated results, showing first 1000 rows.

Command took 0.99 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:36:11 AM on My Cluster

Exists

Exists operates on arrays containing numeric data just the same as those with text data. Let's say that we want to flag the records whose temperatures have exceeded a given value. Run the cell below to view the example.

Cmd 12

```
SELECT
  temps,
  EXISTS(temps, t -> t > 23) highTempsFlag
FROM DeviceData2
```

▸ (1) Spark Jobs

	temps	highTempsFlag
1	▸ [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12]	false
2	▸ [26, 17, 19, 13, 9, 12, 10, 12, 1, 13, 16, 12]	true
3	▸ [11, 13, 19, 8, 14, 16, 13, 14, 14, 9, 7, 12]	false
4	▸ [20, 18, 20, 18, 11, 14, 17, 24, 17, 15, 19, 22]	true

Truncated results, showing first 1000 rows.

Command took 1.15 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:36:16 AM on My Cluster

Transform

When using `TRANSFORM` with numeric data, we can apply any built-in function meant to work with a single value or we can name our own set of operations to be applied to each value in the array. This data includes temperature readings taken in Celsius. Each row contains an array of 12 temperature readings. We can use `TRANSFORM` to convert each element of each array to Fahrenheit. To convert from Celsius to Fahrenheit, multiply the temperature in Celsius by 9, divide by 5, and then add 32.

Let's dissect the code below to better understand the function:

```
TRANSFORM(temps, t -> ((t * 9) div 5) + 32 ) temps_F
```

`TRANSFORM` : the name of the higher-order function
`temps` : the name of our input array
`t` : the name of the iterator variable. You choose this name and then use it in the lambda function. It iterates over the array, cycling each value into the function one at a time.
`->` : Indicates the start of the function
`((t * 9) div 5) + 32` : This is the function. For each value in the input array, the value is multipled by 9 and then divided by 5. Then, we add 32. This is the formula for converting from Celcius to Fahrenheit. Recall that TRANSFORM takes an array, an iterator, and an anonymous function as input. In the code below, temps is the column that holds the array and t is the iterator that cycles through the list. The anonymous function ((t * 9) div 5) + 32 will be applied to each value in the list.

We use the function **`div`** to divide one expression by another without including decimal values. This is strictly for neatness in this example. For operations where precision counts, you would use the `/` operator, which performs floating point division.

Cmd 14

```
SELECT
  temps temps_C,
  TRANSFORM (temps, t -> ((t * 9) div 5) + 32 ) temps_F
FROM DeviceData2;
```

▸ (1) Spark Jobs

	temps_C	temps_F
1	▸ [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12]	▸ [60, 55, 66, 51, 48, 73, 64, 55, 64, 62, 53, 53]
2	▸ [26, 17, 19, 13, 9, 12, 10, 12, 1, 13, 16, 12]	▸ [78, 62, 66, 55, 48, 53, 50, 53, 33, 55, 60, 53]
3	▸ [11, 13, 19, 8, 14, 16, 13, 14, 14, 9, 7, 12]	▸ [51, 55, 66, 46, 57, 60, 55, 57, 57, 48, 44, 53]

Reduce

`REDUCE` is more advanced than `TRANSFORM`; it takes two lambda functions. You can use it to reduce the elements of an array to a single value by merging the elements into a buffer, and applying a finishing function on the final buffer.

We will use the reduce function to find an average value, by day, for our CO_2 readings. Take a closer look at the individual pieces of the `REDUCE` function by reviewing the list below.

```
REDUCE(co2_level, 0, (c, acc) -> c + acc, acc ->(acc div size(co2_level)))
```

`co2_level` is the input array
`0` is the starting point for the buffer. Remember, we have to hold a temporary buffer value each time a new value is added to from the array; we start at zero in this case to get an accurate sum of the values in the list.
`(c, acc)` is the list of arguments we'll use for this function. It may be helpful to think of `acc` as the buffer value and `c` as the value that gets added to the buffer.
`c + acc` is the buffer function. As the function iterates over the list, it holds the total (`acc`) and adds the next value in the list (`c`).
`acc div size(co2_level)` is the finishing function. Once we have the sum of all numbers in the array, we divide by the number of elements to find the average.

This view will be useful in subsequent exercises, so we'll create a temporary view in this step so that we can access it later on.

Cmd 16

```
CREATE OR REPLACE TEMPORARY VIEW Co2LevelsTemporary
AS
  SELECT
    dc_id,
    device_type,
    co2Level,
    REDUCE(co2Level, 0, (c, acc) -> c + acc, acc ->(acc div size(co2Level))) as averageCo2Level

  FROM DeviceData2
  SORT BY averageCo2Level DESC;

SELECT * FROM Co2LevelsTemporary
```

▸ (1) Spark Jobs

	dc_id	device_type	co2Level	averageCo2Level
1	dc-103	sensor-istick	▸ [1819, 1705, 1658, 1753, 1616, 1871]	1737
2	dc-103	sensor-ipad	▸ [1617, 1607, 1835, 1783, 1726, 1568]	1689

Other higher-order functions

There are many built-in functions designed to work with array type data and well as other higher-order functions to explore. You can import this notebook for a list of examples.

Cmd 18

Pivot tables: Example 1

Pivot tables are supported in Spark SQL. A pivot table allows you to transform rows into columns and group by any data field. Let's take a closer look at our query.

`SELECT * FROM ()`: The `SELECT` statement inside the parentheses in the input for this table. Note that it takes two columns from the view `Co2LevelsTemporary`
`PIVOT`: The first argument in the clause is an aggregate function and the column to be aggregated. Then, we specify the pivot column in the `FOR` subclause. The `IN` operator contains the pivot column values.

Cmd 19

```
SELECT * FROM (
  SELECT device_type, averageCo2Level
  FROM Co2LevelsTemporary
)
PIVOT (
  ROUND(AVG(averageCo2Level), 2) avg_co2
  FOR device_type IN ('sensor-ipad', 'sensor-inest',
    'sensor-istick', 'sensor-igauge')
  );
```

▸ (3) Spark Jobs

	sensor-ipad	sensor-inest	sensor-istick	sensor-igauge
1	1245.98	1250.41	1244.86	1247.56

Showing all 1 rows.

Command took 3.25 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:39:26 AM on My Cluster

Pivot Tables: Example 2

In this example, we again pull data from our larger table `DeviceData`. Within the subquery, we create the `month` column and use the `REDUCE` function to create the `averageCo2Level` column.

In the pivot, we take the average of of the `averageCo2Level` values grouped by month. Notice that we rename the month columns from their number to the english abbreviations.

Learn more about pivot tables in this blog post.

Cmd 21

```
SELECT
  *
FROM
  (
    SELECT
      month(date) month,
      REDUCE(co2Level, 0, (c, acc) -> c + acc, acc ->(acc div size(co2Level))) averageCo2Level
    FROM
      DeviceData2
  ) PIVOT (
    avg(averageCo2Level) avg FOR month IN (7 JUL, 8 AUG, 9 SEPT, 10 OCT, 11 NOV)
  )
```

▸ (3) Spark Jobs

	JUL	AUG	SEPT	OCT	NOV
1	1242.8850806451612	1250.8649193548388	1245.1229166666667	1249.2983870967741	1247.7875

Showing all 1 rows.

Command took 2.13 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:40:10 AM on My Cluster

Rollups

Rollups are operators used with the `GROUP BY` clause. They allow you to summarize data based on the columns passed to the `ROLLUP` operator.

This results of this query include average CO_2 levels, grouped by `dc_id` and `device_type`. Rollups are creating subtotals for a specific group of data. The subtotals introduce new rows, and the new rows will contain `NULL` values for everything other than the calculated subtotal.

We can alter this by using the `COALESCE()` function introduced in a previous lesson.

Cmd 23

```
SELECT
   COALESCE(dc_id, "All data centers") AS dc_id,
   COALESCE(device_type, "All devices") AS device_type,
   ROUND(AVG(averageCo2Level))  AS avgCo2Level
FROM Co2LevelsTemporary
GROUP BY ROLLUP (dc_id, device_type)
ORDER BY dc_id, device_type;
```

▸ (2) Spark Jobs

	dc_id	device_type	avgCo2Level
1	All data centers	All devices	1247
2	dc-101	All devices	1197
3	dc-101	sensor-igauge	1202
4	dc-101	sensor-inest	1197
5	dc-101	sensor-ipad	1194
6	dc-101	sensor-istick	1196
7	dc-102	All devices	1296

Showing all 21 rows.

Command took 3.88 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:40:50 AM on My Cluster

Cube

`CUBE` is also an operator used with the `GROUP BY` clause. Similar to `ROLLUP`, you can use `CUBE` to generate summary values for sub-elements grouped by column value. `CUBE` is different than `ROLLUP` in that it will also generate subtotals for all combinations of grouping columns specified in the `GROUP BY` clause.

Notice that the output for the example below shows some of additional values generated in this query. Data from `"All data centers"` has been aggregated across device types for all centers.

Cmd 25

```
SELECT
   COALESCE(dc_id, "All data centers") AS dc_id,
   COALESCE(device_type, "All devices") AS device_type,
   ROUND(AVG(averageCo2Level))  AS avgCo2Level
FROM Co2LevelsTemporary
GROUP BY CUBE (dc_id, device_type)
ORDER BY dc_id, device_type;
```

▸ (2) Spark Jobs

	dc_id	device_type	avgCo2Level
1	All data centers	All devices	1247
2	All data centers	sensor-igauge	1248
3	All data centers	sensor-inest	1250
4	All data centers	sensor-ipad	1246
5	All data centers	sensor-istick	1245
6	dc-101	All devices	1197
7	dc-101	sensor-igauge	1202

Showing all 25 rows.

Command took 1.49 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:41:32 AM on My Cluster

Cmd 26

```
%run ../Includes/Classroom-Cleanup

```

Command took 0.04 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 9:41:37 AM on My Cluster

Partitioning Tables

You can affect query performance by partitioning data in your tables. Recall that we loooked at some specific performance improvements (and downgrades) that can be caused by partitioning in Module 4, Spark Under the Hood. Partitoning data in a Spark SQL query creates a subdirectory of data accoording to a rule. For example, if I partition a set of data by year, all data in any given folder in the subdirectories for that table will have the same year. That means, when is comes time to query the set and I include something like,

`WHERE year = 1990`,

Spark can avoid reading any data from folders that are **not** in the `1990` subfolder.

In the next set of exercises,we will demonstrate examples of how to partition data, how to view table partitions, and how to use widgets to adjust your query parameters.

Finally, we will demonstrate how to use window fucntions, which use a different sort of partitioning to compute values over a sub-section of a table.

Run the cell below to set up your classroom environment.

Cmd 3

```
%run ../Includes/6-3-Setup
```

In the example below, we create and use a table called `AvgTemps`. You may recognize this query from a previous notebook. This table includes temperature readings taken over entire days as well as the calculated value `avg_daily_temp_c`.

Notice that this table has been `PARTITIONED BY` the column `device_type`. The result of this kind of partitioning is that the table is stored in separate files. This may speed up subsequent queries that can filter out certain partitions. These are **not** the same partitions we refer to when discussing basic Spark architecture.

⚠️ The word **partition** is a bit overloaded in big data and distributed computing. That is, we have to pay careful attention to the context to understand what sort of partition is being discussed. In particular, when referring to Spark's internal architecture, partitions refer to the units of data that are phyiscally distributed across a cluster. Managing this kind of partitioning is generally the job of a data engineer and is outside of the scope of this course.

Cmd 5

```
CREATE TABLE IF NOT EXISTS AvgTemps
PARTITIONED BY (device_type)
AS
  SELECT
    dc_id,
    date,
    temps,
    REDUCE(temps, 0, (t, acc) -> t + acc, acc ->(acc div size(temps))) as avg_daily_temp_c,
    device_type
  FROM DeviceData2;

SELECT * FROM AvgTemps;
```

▸ (6) Spark Jobs

	dc_id	date	temps	avg_daily_temp_c	device_type
1	dc-101	2019-07-01	▸ [16, 13, 19, 11, 9, 23, 18, 13, 18, 17, 12, 12]	15	sensor-igauge
2	dc-101	2019-07-02	▸ [3, 1, 7, 8, 2, -4, 7, 1, 0, 8, -2, 3]	2	sensor-igauge
3	dc-101	2019-07-03	▸ [16, 14, 10, 12, 17, 10, 13, 11, 8, 19, 11, 21]	13	sensor-igauge
4	dc-101	2019-07-04	▸ [16, 20, 15, 22, 18, 15, 12, 18, 22, 18, 20, 20]	18	sensor-igauge
5	dc-101	2019-07-05	▸ [8, 7, 13, 2, 5, 6, 6, 7, 9, 3, 2, 3]	5	sensor-igauge

Truncated results, showing first 1000 rows.

Command took 19.35 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:06:59 AM on My Cluster

Show partitions

Use the command `SHOW PARTITIONS` to see how your data is partitioned. In this case, we can verify that the data has been partitioned according to device type.

Cmd 7

```
SHOW PARTITIONS AvgTemps
```

▸ (2) Spark Jobs

	device_type
1	sensor-ipad
2	sensor-inest
3	sensor-istick
4	sensor-igauge

Showing all 4 rows.

Command took 1.97 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:08:41 AM on My Cluster

Create a widget

Input widgets allow you to add parameters to your notebooks and dashboards. You can create and remove widgets, as well as retrieve values from them within a SQL query. Once created, they appear at the top of your notebook. You can design them to take user input as a:

- dropdown: provide a list of options for the user to select from
- text: user enters input as text
- combobox: Combination of text and dropdown. User selects a value from a provided list or input one in the text box.
- multiselect: Select one or more values from a list of provided values

Widgets are best for:

- Building a notebook or dashboard that is re-executed with different parameters
- Quickly exploring results of a single query with different parameters

Learn more about widgets here.

We have already created a partitioned table, so we have one designated column meant to be used for easy data reads with filters. In this example, we'll use a widget to allow anyone viewing the notebook (or correspoding dashboard) the ability to filter by `device_type` in our table.

In this example, we use a `DROPDOWN` so that the user can select among all available options. We name the widget `selectedDeviceType` and specify the `CHOICES` by getting a distinct list of all values in the `deviceType` column.

Cmd 9

```
CREATE WIDGET DROPDOWN selectedDeviceType DEFAULT "sensor-inest" CHOICES
SELECT
  DISTINCT device_type
FROM
  DeviceData2
```

▸ (2) Spark Jobs

OK

Command took 0.99 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:09:01 AM on My Cluster

Use the selected value in your query

We use a user-defined function, `getArgument()` to retrieve the current value selected in the widget. This functionality is available in the Databricks Runtime, but not open-source Spark.

In the example below, we retrieve the selected value in the `WHERE` clause at the bottom of the query. Run the example. Then, change the value in the widget. Notice that the command below runs automatically. By default, cells that access input from a given widget will rerun automatically when the input value is changed. You can change default values using the ⚙ icon on the right side of the widgets panel at the top of the notebook.

Cmd 11

```
SELECT
  device_type,
  ROUND(AVG(avg_daily_temp_c),4) AS avgTemp,
  ROUND(STD(avg_daily_temp_c), 2) AS stdTemp
FROM AvgTemps
WHERE device_type = getArgument("selectedDeviceType")
GROUP BY device_type
```

▸ (2) Spark Jobs

	device_type	avgTemp	stdTemp
1	sensor-inest	15.4804	4.25

Showing all 1 rows.

Command took 1.38 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:09:08 AM on My Cluster

Remove widget

You can remove widget with the following command by simply referencing it by name.

Cmd 13

```
REMOVE WIDGET selectedDeviceType
```

OK

Command took 0.02 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:12:29 AM on My Cluster

Cmd 14

Window functions

Window functions calculate a return variable for every input row of a table based on a group of rows selected by the user, the frame. To use window functions, we need to mark that a function is used as a window by adding an `OVER` clause after a supported function in SQL. Within the `OVER` clause, you specify which rows are included in the frame associated with this window.

In the example, the function we will use is `AVG` . We define the Window Specification associated with this function with `OVER(PARTITION BY ...)` . The results show that the average monthly temperature is calculated for a data center on a given date. The `WHERE` clause at the end of this query is included to show a whole month of data from a single data center.

Cmd 15

```
SELECT
  dc_id,
  month(date),
  avg_daily_temp_c,
  AVG(avg_daily_temp_c)
  OVER (PARTITION BY month(date), dc_id) AS avg_monthly_temp_c
FROM AvgTemps
WHERE month(date)="8" AND dc_id = "dc-102";
```

▸ (3) Spark Jobs

	dc_id	month(date)	avg_daily_temp_c	avg_monthly_temp_c
1	dc-102	8	21	16.870967741935484
2	dc-102	8	15	16.870967741935484
3	dc-102	8	16	16.870967741935484
4	dc-102	8	15	16.870967741935484
5	dc-102	8	13	16.870967741935484

CTEs with window functions

Here, we integrate a that same window functionand use a CTE to further manipulate values calculated in the common table expression.

Cmd 17

```
1  WITH DiffChart AS
2  (
3  SELECT
4    dc_id,
5    date,
6    avg_daily_temp_c,
7    AVG(avg_daily_temp_c)
8    OVER (PARTITION BY month(date), dc_id) AS avg_monthly_temp_c
9  FROM AvgTemps
10 )
11 SELECT
12   dc_id,
13   date,
14   avg_daily_temp_c,
15   avg_monthly_temp_c,
16   avg_daily_temp_c - ROUND(avg_monthly_temp_c) AS degree_diff
17 FROM DiffChart;
```

▸ (2) Spark Jobs

	dc_id	date	avg_daily_temp_c	avg_monthly_temp_c	degree_diff
1	dc-101	2019-07-01	15	14.798387096774194	0
2	dc-101	2019-07-02	2	14.798387096774194	-13
3	dc-101	2019-07-03	13	14.798387096774194	-2
4	dc-101	2019-07-04	18	14.798387096774194	3
5	dc-101	2019-07-05	5	14.798387096774194	-10
6	dc-101	2019-07-06	11	14.798387096774194	-4
7	dc-101	2019-07-07	17	14.798387096774194	2

Truncated results, showing first 1000 rows.

Command took 2.30 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:13:33 AM on My Cluster

```sql
WITH DiffChart AS
(
SELECT
  dc_id,
  date,
  avg_daily_temp_c,
  AVG(avg_daily_temp_c)
  OVER (PARTITION BY month(date), dc_id) AS avg_monthly_temp_c
FROM AvgTemps
)
SELECT
  dc_id,
  date,
  avg_daily_temp_c,
  avg_monthly_temp_c,
  avg_daily_temp_c - ROUND(avg_monthly_temp_c) AS degree_diff
FROM DiffChart;
```

▸ (5) Spark Jobs

Aggregated (by avg) in the backend.

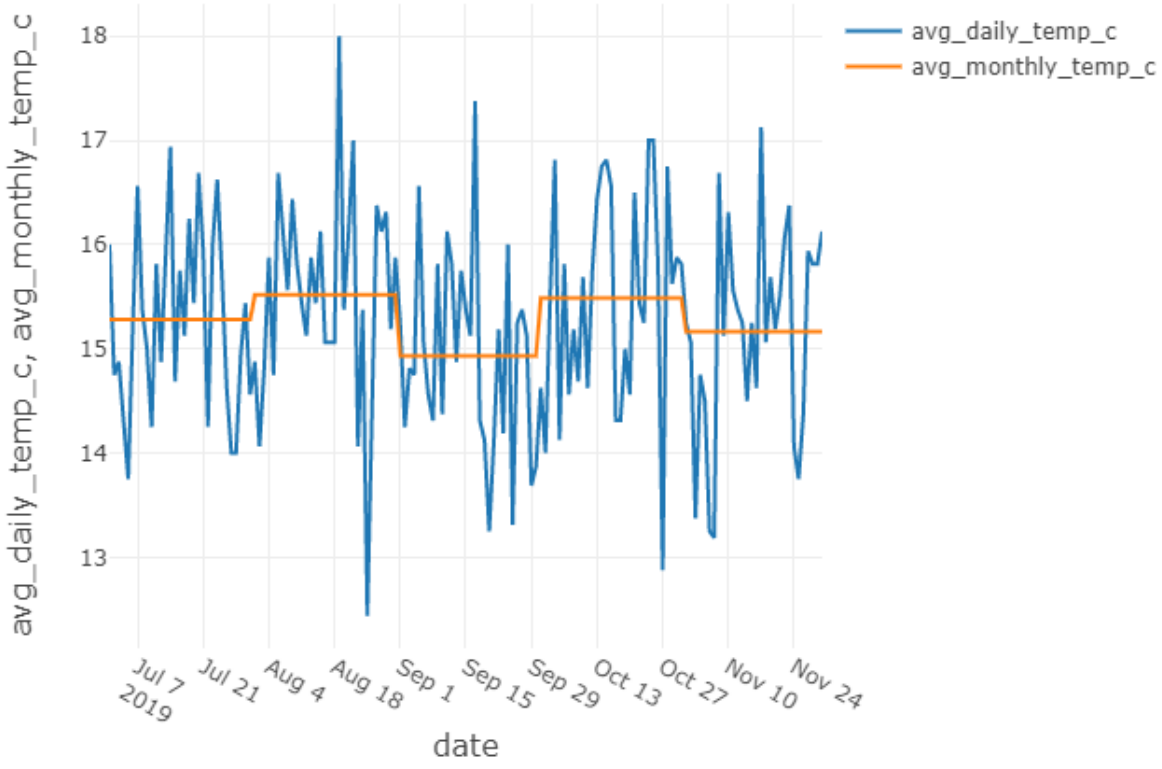
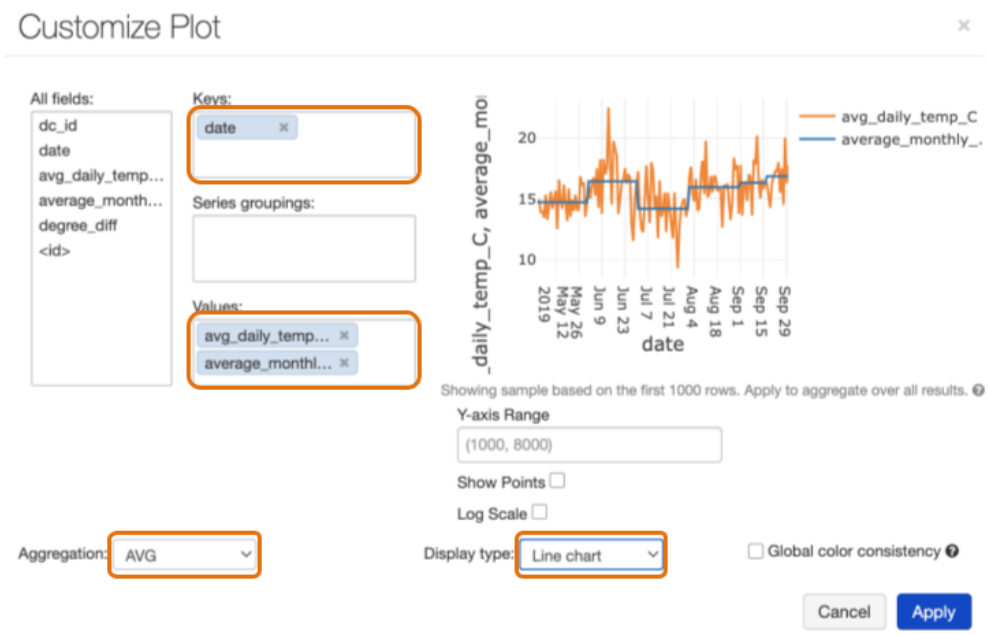


Chart results

First, use the graph tool to create the visualization. Configure your `Plot Options` to match the selections in the image below.



In the `Keys:` dialog box, add `date`.
In the `Values:` dialog box, add `avg_daily_temp_c` and `avg_monthly_temp_c`.
The `Aggregation` value should be set to `AVG` and the `Display type` set to `Line Chart`.

Cmd 19

```
%run ../Includes/Classroom-Cleanup
```

Lab 3 - Sharing Insights

Module 6 Assignment

In this lab, we will explore a small mock data set from a group of data centers. You'll see that is is similar to the data you have been working with, but it contains a few new columns and it is structured slightly differently to test your skills with hierarchical data manipulation.

In this assignment you will:

- Apply higher-order functions to array data
- Apply advanced aggregation and summary techniques to process data
- Present data in an interactive dashboard or static file

As you work through the following tasks, you will be prompted to enter selected answers in Coursera. Find the quiz associated with this lab to enter your answers.

Run the cell below to prepare this workspace for the lab.

Cmd 3

```
%run ../Includes/Classroom-Setup
```

Command took 5.71 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:34:50 AM on My Cluster

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = false
res2: Boolean = false
```

Exercise 1: Create a table

Summary: Create a table.

Use this path to access the data: `/mnt/training/iot-devices/data-centers/energy.json`

Steps to complete:

- Write a `CREATE TABLE` statement for the data located at the endpoint listed above
- Use json as the file format

Cmd 5

```
1  Drop table if exists Energy;
2  Create table Energy
3    Using json
4    Options (
5    path "/mnt/training/iot-devices/data-centers/energy.json"
6    );
7
8  Describe Energy;
```

▸ (1) Spark Jobs

	col_name	data_type	comment
1	battery_level	array<bigint>	null
2	co2_level	array<bigint>	null
3	device_id	bigint	null
4	device_type	string	null
5	signal	array<bigint>	null
6	temps	array<bigint>	null
7	timestamp	string	null

Showing all 7 rows.

Command took 5.13 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 11:09:10 PM on My Cluster

Exercise 2: Sample the table

Summary: Sample the table to get a closer look at a few rows

Steps to complete:

- Write a query that allows you to see a few rows of the data

Cmd 7

```
1  Select * from Energy limit 5
```

▸ (1) Spark Jobs

	battery_level	co2_level	device_id	device_type	signal	temps	timestamp
1	▸ [3, 3, 2]	▸ [1343, 1595, 1405]	0	sensor-istick	▸ [24, 24, 25]	▸ [22, 23, 21, 23]	2019/08/02 15:00:00
2	▸ [1, 1, 2]	▸ [1213, 1346, 1247]	1	sensor-inest	▸ [22, 24, 24]	▸ [22, 37, 34, 39]	2019/07/01 03:00:00
3	▸ [3, 1, 3]	▸ [1261, 1216, 1258]	2	sensor-ipad	▸ [25, 24, 28]	▸ [32, 33, 40, 44]	2019/06/03 11:00:00
	▸ [4, 3, 4]	▸ [1270, 1257	3	sensor-igauge	▸ [24, 22, 24]	▸ [22, 13, 18	2019/09/21 07:00:00

Showing all 5 rows.

Command took 2.86 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 10:41:11 PM on My Cluster

Exercise 3: Create view

Summary: Create a temporary view that displays the timestamp column as a timestamp.

Steps to complete:

- Create a temporary view named `DCDevices`
- Convert the `timestamp` column to a timestamp type. Refer to the Datetime patterns documentation for the formatting information.
- (Optional) Rename columns to use camelCase

Cmd 9

```
Create or replace temporary view DCDevices as (
  select
    battery_level as batteryLevel,
    co2_level as co2Level,
    device_id as deviceId,
    device_type as deviceType,
    signal,
    cast(temps as array<int>) as temps,
    timestamp as date
  from Energy
);
describe DCDevices;
select * from DCDevices limit 3;
```

▸ (1) Spark Jobs

	batteryLevel	co2Level	deviceId	deviceType	signal	temps	date
1	▸ [3, 3, 2]	▸ [1343, 1595, 1405]	0	sensor-istick	▸ [24, 24, 25]	▸ [22, 23, 21, 23]	2019/08/02 15:00:00
2	▸ [1, 1, 2]	▸ [1213, 1346, 1247]	1	sensor-inest	▸ [22, 24, 24]	▸ [22, 37, 34, 39]	2019/07/01 03:00:00

Showing all 3 rows.

Command took 0.64 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 11:42:10 PM on My Cluster

Exercise 4: Flag records with defective batteries

Summary: When a battery is malfunctioning, it can report negative battery levels. Create a new boolean column `needService` that shows whether a device needs service.

Steps to complete:

- Write a query that shows which devices have malfunctioning batteries
- Include columns `batteryLevel`, `deviceId`, and `needService`
- Order the results by `deviceId`, and then `batteryLevel`
- **Answer the corresponding question in Coursera**

Cmd 11

```
Select batteryLevel, deviceId,
  EXISTS (batteryLevel, batteryLevel -> batteryLevel < 0) needService
from DCDevices
order by deviceId, batteryLevel
```

▸ (1) Spark Jobs

	batteryLevel	deviceId	needService
1	▸ [-4, -1, -1]	0	true
2	▸ [-4, -1, -1]	0	true
3	▸ [-4, -1, 0]	0	true
4	▸ [-3, -3, -4]	0	true
5	▸ [-3, -2, -2]	0	true
6	▸ [-3, -2, -2]	0	true
7	▸ [-3, -2, -2]	0	true

Truncated results, showing first 1000 rows.

Command took 4.20 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 11:11:32 PM on My Cluster

Exercise 5: Display high CO_2 levels

Summary: Create a new column to display only CO_2 levels that exceed 1400 ppm.

Steps to complete:

- Include columns `deviceId`, `deviceType`, `highCO2`, `time`
- The column `highCO2` should contain an array of CO_2 readings over 1400
- Show only records that contain `highCO2` values
- Order by `deviceId`, and then `highCO2`

Answer the corresponding question in Coursera

 You may need to use a subquery to write this in a single query statement.

Cmd 13

```
1  select *
2  from (
3    select deviceId, deviceType, co2Level,
4      FILTER (co2Level, co2Level -> co2Level > 1400) highCO2,
5      date
6    from DCDevices
7    )
8  where size(highCO2) > 0
9  order by deviceId, highCO2
```

▸ (1) Spark Jobs

	deviceId	deviceType	co2Level	highCO2	date
1	0	sensor-istick	▸ [1336, 1401, 1353]	▸ [1401]	2019/08/16 21:00:00
2	0	sensor-ipad	▸ [1321, 1401, 1281]	▸ [1401]	2019/06/29 10:00:00
3	0	sensor-ipad	▸ [1401, 1235, 1245]	▸ [1401]	2019/08/12 12:00:00

Exercise 6: Create a partitioned table

Summary: Create a new table partitioned by `deviceId`

Steps to complete:

- Include all columns
- Create the table using Parquet
- Rename the partitioned column `p_deviceId`
- Run a `SELECT *` to view your table.

Answer the corresponding question in Coursera

Cmd 15

```
Drop table if exists Device;
Create Table Device
Partitioned by (p_deviceId)
as Select batteryLevel, co2Level, deviceType, signal, temps, date,
    REDUCE(temps, 0, (t, acc) -> t + acc, acc -> (acc div size(temps))) as avg_temps,
    deviceId as p_deviceId
   From DCDevices
;
select * from device;
```

▸ (6) Spark Jobs

	batteryLevel	co2Level	deviceType	signal	temps	date	avg_temps	p_deviceId
1	▸ [1, 1, 2]	▸ [1213, 1346, 1247]	sensor-inest	▸ [22, 24, 24]	▸ [22, 37, 34, 39]	2019/07/01 03:00:00	33	1
2	▸ [7, 6, 7]	▸ [1352, 1336, 1376]	sensor-inest	▸ [23, 21, 24]	▸ [17, 24, 28, 23]	2019/09/04 16:00:00	23	1
3	▸ [-3, -1, -1]	▸ [1032, 1170, 1167]	sensor-istick	▸ [21, 21, 21]	▸ [20, 12, 23, 26]	2019/06/13 17:00:00	20	1
4	▸ [5, 6, 6]	▸ [1117, 1113, 968]	sensor-istick	▸ [24, 25, 24]	▸ [29, 32, 31, 29]	2019/09/30 13:00:00	30	1

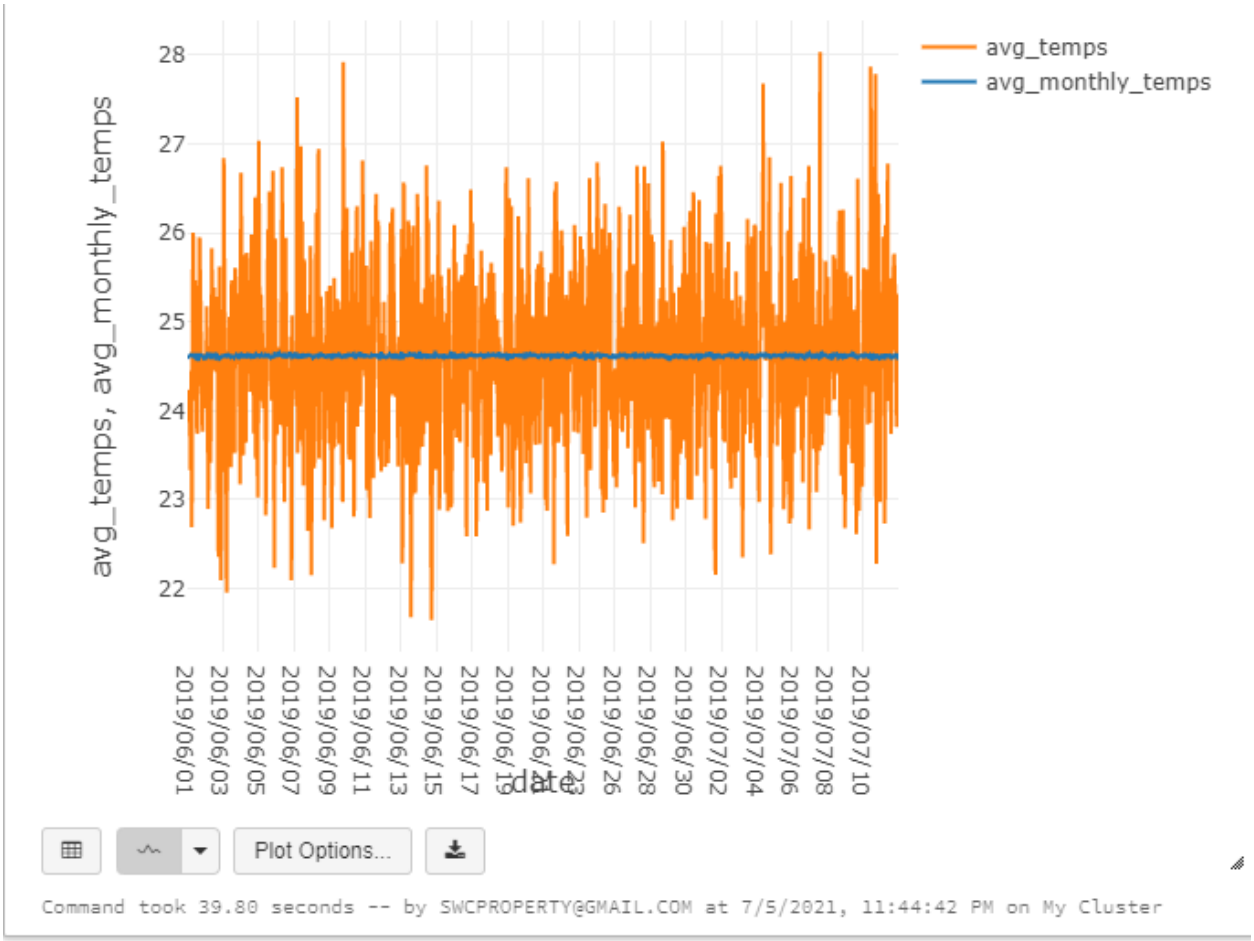
Exercise 7: Visualize average temperatures

Cmd 17

```
1  With DiffChart as
2  (
3    Select
4      p_deviceId,
5      date,
6      avg_temps,
7      AVG(avg_temps)
8      OVER (PARTITION BY month(date), p_deviceId) as avg_monthly_temps
9    From
10     Device
11 )
12 Select
13   p_deviceId,
14   date,
15   avg_temps,
16   avg_monthly_temps,
17   avg_temps - round(avg_monthly_temps) as temp_difference
18 From
19   DiffChart;
```

▸ (5) Spark Jobs

Aggregated (by avg) in the backend.
Truncated results, showing first 1000 rows.



Command took 39.80 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 11:44:42 PM on My Cluster

Cmd 18

Exercise 8: Create a widget

Cmd 19

```
1  Create WIDGET DROPDOWN selectedDeviceId DEFAULT "0" CHOICES
2  Select
3    Distinct p_deviceId
4  From
5    Device;
```

Exercise 9: Use the widget in a query

Cmd 21

```
Select
  p_deviceId,
  Round(avg(avg_temps),4) as avg_temp,
  Round(std(avg_temps),2) as std_temp
From
  Device
Where
  p_deviceId = getArgument("selectedDeviceId")
Group by p_deviceId;
```

▸ (2) Spark Jobs

	p_deviceId	avg_temp	std_temp
1	0	24.5102	6.39

Showing all 1 rows.

Command took 2.27 seconds -- by SWCPROPERTY@GMAIL.COM at 7/5/2021, 11:51:51 PM on My Cluster

Cmd 22

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

Big data characteristics

Data Warehouses

First appearing in the 1980s, data warehouses evolved as big data emerged and became one of the first solutions to accommodate its needs. Before the data warehouse, companies were storing data in lots of different systems with no way to unite the disparate databases. For analysts, a disparate system can make it impossible to synthesize a total view on which to base actionable business insights.

Data warehouses brought the collection of databases all under a single umbrella and allowed the data to be queried and viewed as a whole.

Advantages

In the late 1990's, data warehouses were the most dominant data architecture for big companies. The primary advantages include:

- Standard structured query language (SQL) for access
- Integration of many data sources
- Data optimized for fast reads
- Ability to run quick ad-hoc analytical queries

These advantages enable data analysts to use an intuitive and powerful query language, SQL, to easily access data from a variety of sources. Also, the data is stored such that it can be delivered quickly to show results of analytical queries.

Challenges

As the speed and scale of data really exploded though, some deficiencies became clear:

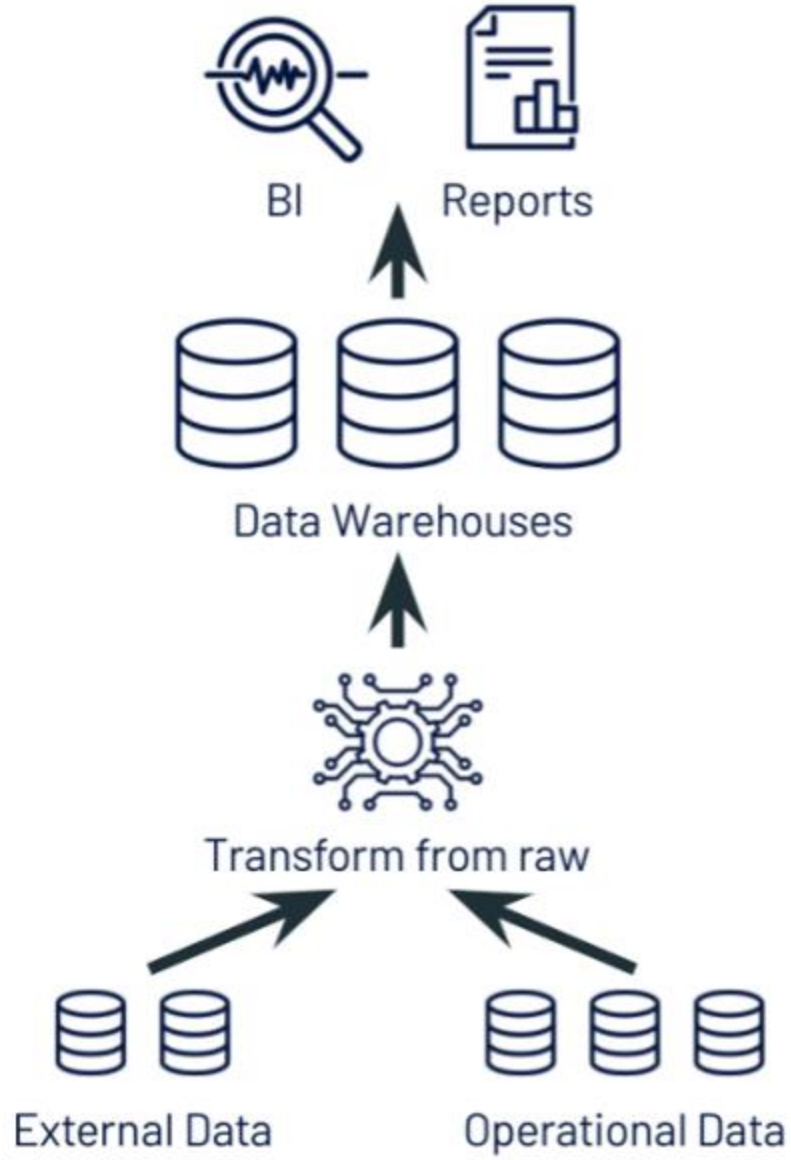
- Inability to store unstructured, raw data
- Difficult to scale
- Requires significant investment in a proprietary system

All of these add up to bottlenecks and roadblocks for data analysts.

Since warehouses are unable to store unstructured, raw data, data teams who collect both are automatically resigned to work within different systems. As big data gets bigger, it continues to push the limits of a legacy data warehouse. Scaling up requires engineers dedicated strictly to managing the infrastructure. More time on systems engineering translates into delays in analytics.

Once a company builds expensive, proprietary hardware and software into their system, it can be almost impossible to leave it.

Now, there are some cloud-based data warehouse solutions that have been built to address some of these challenges, but still - we are missing support for unstructured data and streaming data. As a result, many organizations employ data warehouses only for smaller subsets of their data, as in the diagram below. You can see that much of their data is flowing into data storage outside of the data warehouse, and only subsets are available at the data warehouse level.



Data Lakes

Data Lakes are newer to the landscape, emerging around 2010 and developing over the past decade, to answer some of the problems organizations were facing with their data warehouses. As big data continues to evolve, the data we collect is increasingly unstructured, fast-moving, and high-volume.

Data lakes are often used to consolidate all of an organization’s data in a single, central location, where it can be saved “as is,” without the need to impose a schema or structure on it up front. Data in all stages of the refinement process can be stored in a data lake: raw data can be ingested and stored right alongside an organization’s structured, tabular data sources (like database tables), as well as intermediate data tables generated in the process of refining raw data. Unlike most databases, data lakes can process all data types including images, video, audio and text.

Today, companies have lots of data, but it’s often isolated and siloed away in different storage systems: data warehouses, databases, and other storage systems across the enterprise. A data lake breaks down these data silos, centralizing and consolidating all of your organization’s data assets into a complete and authoritative data store for analytics that is always up to date. Unifying all data in a data lake is the first step for companies that aspire to harness the power of machine learning and data analytics to win in the next decade.

A data lake’s flexible, unified architecture opens up a wide range of new use cases for cross-functional enterprise scale analytics, BI, and machine learning projects that can unlock massive business value. Data analysts can harvest rich insights by querying the data lake using SQL, data scientists can join and enrich data sets to generate ML models with ever greater accuracy, data engineers can build automated ETL pipelines, and business intelligence analysts can create visual dashboards and reporting tools faster and easier than before. These use cases can all be performed on the data lake simultaneously, without lifting and shifting the data, even while new data is streaming in.

Advantages

- Can hold all of an organization's data: structured, unstructured, and semi-structured
- Centralizes data for access by the whole data team
- Storage is relatively inexpensive, and the quantity of storage can be easily increased for scalability

Challenges

- Data reliability - without proper tools, it can be difficult to maintain data lakes which can threaten the veracity of an organizations data.
- Query performance - in many systems, as data lakes grow larger, query performance suffers. This can be due to a variety of factors including slowdowns around metadata management and improper data partitioning.

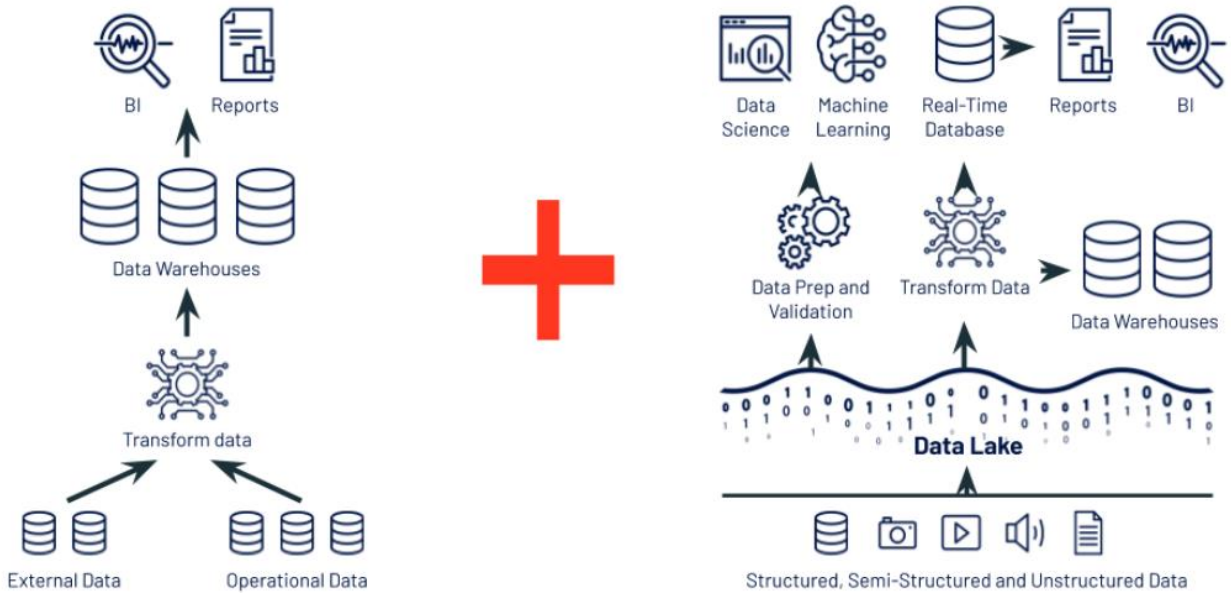
	Data lake	Data warehouse
Primary types of data	All types: Structured data, semi-structured data, unstructured (raw) data	Structured data only
Cost	$	$$$
Scalability	Scales to hold any amount of data at low cost, regardless of type	Scaling up becomes exponentially more expensive due to vendor costs
Intended users	Data analysts, data scientists	Data analysts
Vendor lock-in	No	Yes
Advantages	Low cost, flexibility, scalability, allows storage of the raw data needed for machine learning	User interface is familiar to users of traditional databases
Disadvantages	Exploring large amounts of raw data can be difficult without tools to organize and catalog the data	Expensive, always-on architecture, proprietary software, cannot hold unstructured (raw) data needed for machine learning

The Lakehouse

Over the past few years at Databricks, we've seen a new data management paradigm that emerged independently across many customers and use cases: the lakehouse. In this reading we describe this new paradigm and its advantages over previous approaches.

We've identified both benefits and challenges for two types of popular data storage solutions available for working with big data. The lists of benefits and challenges in each system do tend to align - that is, where one system falters, the other is proficient and vice versa.

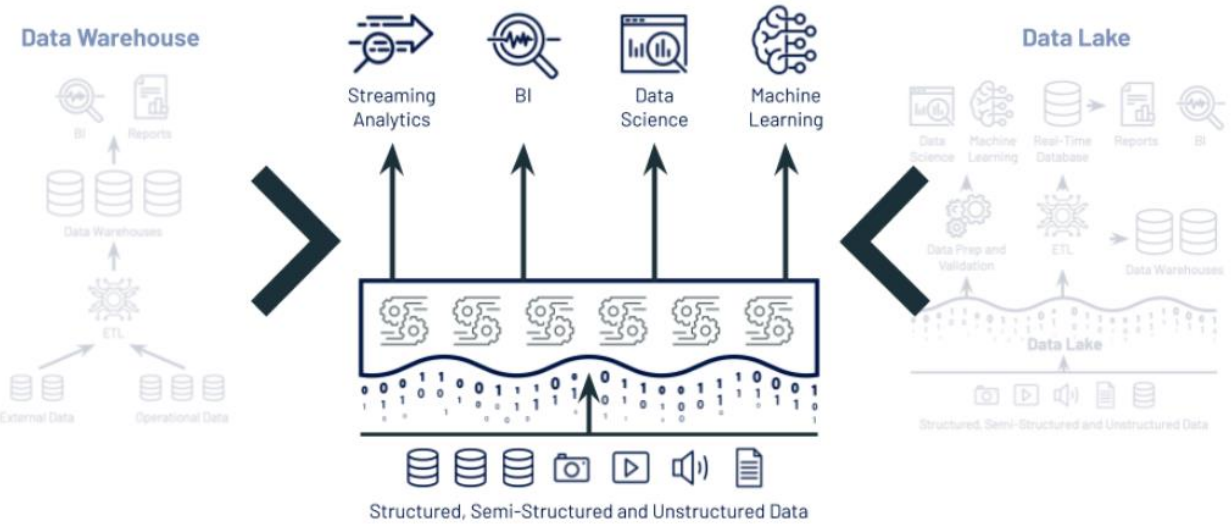
As a result, many organizations have taken to using some form of both, ostensibly hoping to develop a solution that can offer the best from both systems.



Following this path, we quickly find out that maintaining two systems is difficult at best, and at worst impossible. It is high cost, complex, and not reliable as a single source of truth.

The Lakehouse

A Lakehouse combines the best elements of data lakes and data warehouses. They use similar data management features as those at work in data warehouses, thereby delivering the fast query speeds and reliable, manageable data. But, they are built directly on the low-cost and flexible storage used for data lakes. They are what you would get if you had to redesign data warehouses in the modern world, now that cheap and highly reliable storage (in the form of object stores) are available.



A lakehouse has the following key features, among others:

- support for diverse data types and formats
- ability to use BI tools directly on source data
- support for diverse workloads (BI, data science, machine learning, and analytics)
- data reliability and consistency

From BI to AI

The lakehouse is a new data management paradigm that radically simplifies enterprise data infrastructure and accelerates innovation in an age when machine learning is poised to disrupt every industry. In the past most of the data that went into a company's products or decision making was structured data from operational systems, whereas today, many products incorporate AI in the form of computer vision and speech models, text mining, and others. Why use a lakehouse instead of a data lake for AI? A lakehouse gives you data versioning, governance, security and ACID properties that are needed even for unstructured data.

Current lakehouses reduce cost but their performance can still lag specialized systems (such as data warehouses) that have years of investments and real-world deployments behind them. Users may favor certain tools (BI tools, IDEs, notebooks) over others so lakehouses will also need to improve their UX and their connectors to popular tools so they can appeal to a variety of personas. These and other issues will be addressed as the technology continues to mature and develop. Over time lakehouses will close these gaps while retaining the core properties of being simpler, more cost efficient, and more capable of serving diverse data applications.

Adapted from: https://databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html

What is Delta Lake?

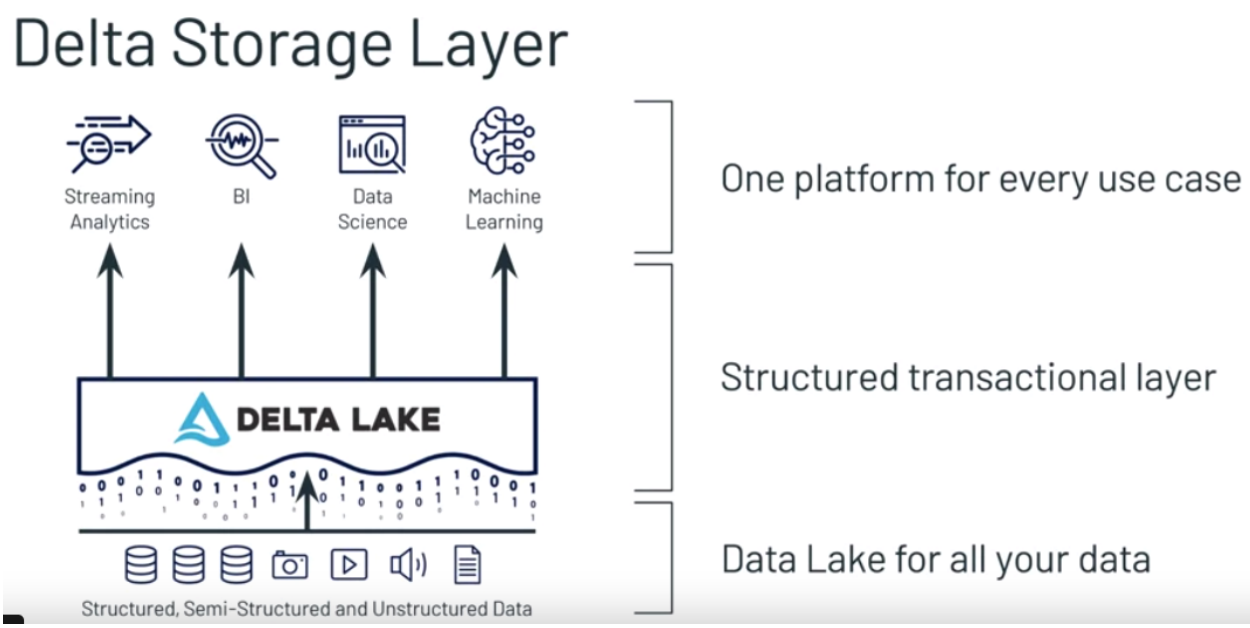
- Core component of a data lakehouse
- Offers guaranteed consistency because it's ACID compliant
- Robust data store
- Designed to work with Apache Spark

databricks

Elements of Delta Lake

- Delta Architecture
- Delta Storage Layer
- Delta Engine
- Delta Tables


Delta architecture
DELTA LAKE
DATA
Bronze
Silver
Gold
Streaming Analytics
AI & Reporting
Data quality

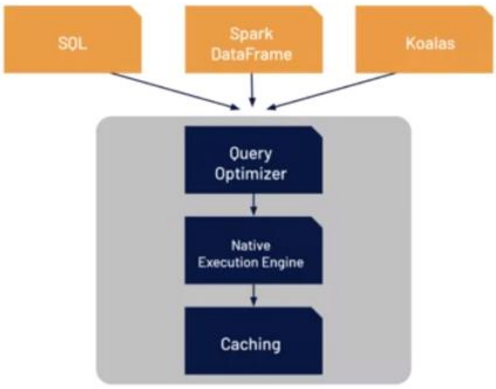

Delta Storage Layer
Streaming Analytics
BI
Data Science
Machine Learning
DELTA LAKE
Structured, Semi-Structured and Unstructured Data
One platform for every use case
Structured transactional layer
Data Lake for all your data

Delta Storage Layer

- Guarantee data is consistent
- Track metadata
- Automatically handle variations in schema
- Enables version control and rollbacks
- Merge and update data as it arrives

Delta Engine

- File management optimizations
- Performance optimization with Delta Caching
- Dynamic File Pruning
- Adaptive Query Execution



Delta tables

- Creates Delta files
- Registers the table in the Metastore
- Starts a transaction log

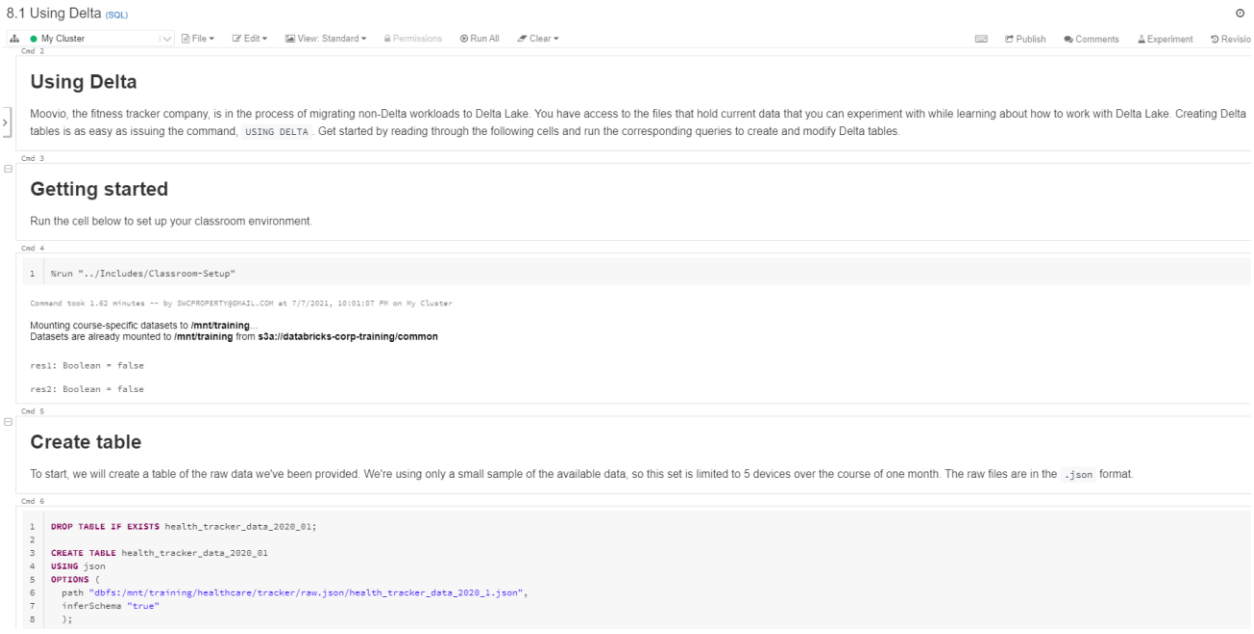
Instead of this:

```
CREATE TABLE events
USING json
AS SELECT *
FROM json.`/data/events/`
```

Write this:

```
CREATE TABLE events
USING delta
AS SELECT *
FROM json.`/data/events/`
```

databricks



Preview data

Before we do anything else, let's quickly inspect the data by viewing a sample.

Cmd 8

```
SELECT * FROM health_tracker_data_2020_01 TABLESAMPLE (5 ROWS)
```

▸ (1) Spark Jobs

	month	value
1	2020-01	▸ {"device_id": 0, "heartrate": 101.3720506847, "name": "Deborah Powell", "time": 1577836800}
2	2020-01	▸ {"device_id": 0, "heartrate": 98.7361247811, "name": "Deborah Powell", "time": 1577840400}
3	2020-01	▸ {"device_id": 0, "heartrate": 99.9724260825, "name": "Deborah Powell", "time": 1577844000}
4	2020-01	▸ {"device_id": 0, "heartrate": 99.8718710286, "name": "Deborah Powell", "time": 1577847600}
5	2020-01	▸ {"device_id": 0, "heartrate": 98.3444848503, "name": "Deborah Powell", "time": 1577851200}

Showing all 5 rows.

Command took 2.56 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:18:51 PM on My Cluster

Cmd 9

Create Delta table

This example, so far, is of a Bronze level table. We can display the raw data, but it is not easily queryable. Our next step is to create a cleaned Silver table. This table may flow into several business aggregate Gold level tables later on. In this step, we'll focus on creating an easily queryable table that includes most or all of the data, with all columns accurately typed and object properties unpacked into individual columns.

Recall that a Delta table consists of three things:

1. The Delta files (in object storage)
2. The Delta Transaction Log saved with the Delta files in object storage.
3. The Delta table registered in the Metastore

Run the cell below to create a new table using `DELTA`. This step registers your table in the metastore, converts your files to Delta and creates the transaction log, which will hold the record of every transaction that is performed on this table.

```
CREATE OR REPLACE TABLE health_tracker_silver
USING DELTA
PARTITIONED BY (p_device_id)
LOCATION "/health_tracker/silver" AS (
SELECT
  value.name,
  value.heartrate,
  CAST(FROM_UNIXTIME(value.time) AS timestamp) AS time,
  CAST(FROM_UNIXTIME(value.time) AS DATE) AS dte,
  value.device_id p_device_id
FROM
  health_tracker_data_2020_01
)

```

▸ (4) Spark Jobs

Query returned no results

Command took 24.74 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:19:00 PM on My Cluster

Cmd 11

Great! You have created your first Delta table! Run the `DESCRIBE DETAIL` command to view table details. You can see that the table format is `delta` and it is stored in the location you specified.

Cmd 12

```
DESCRIBE DETAIL health_tracker_silver
```

	format	id	name	description	location	createdAt	lastModified	partitionColumns	numFiles	sizeInBytes
1	delta	e4a1fa0b-1cbe-4a8c-9375-3e6abace3322	default.health_tracker_silver	null	dbfs:/health_tracker/silver	2021-07-08T05:19:01.676+0000	2021-07-08T05:19:14.000+0000	▸ ["p_device_id"]	5	57263

Showing all 1 rows.

Command took 0.85 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:19:29 PM on My Cluster

Read in new data

Recall that we created that table with just one month of data. Now let's see how we can add new data to that table.

Run the cell below to read in the new raw file.

Cmd 14

```
DROP TABLE IF EXISTS health_tracker_data_2020_02;

CREATE TABLE health_tracker_data_2020_02
USING json
OPTIONS (
  path "dbfs:/mnt/training/healthcare/tracker/raw.json/health_tracker_data_2020_2.json",
  inferSchema "true"
  );
```

▸ (1) Spark Jobs

OK

Command took 1.26 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:19:57 PM on My Cluster

Cmd 15

Append files

We can append the next month of of records to the existing table using the `INSERT INTO` command. We will transform the new data to match the existing schema.

Cmd 16

```
INSERT INTO
  health_tracker_silver
SELECT
  value.name,
  value.heartrate,
  CAST(FROM_UNIXTIME(value.time) AS timestamp) AS time,
  CAST(FROM_UNIXTIME(value.time) AS DATE) AS dte,
  value.device_id p_device_id
FROM
  health_tracker_data_2020_02
```

	num_affected_rows	num_inserted_rows
1	3408	3408

Showing all 1 rows.

Command took 8.94 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:20:04 PM on My Cluster

Cmd 17

Time Travel: Count records in the previous table

Let's count the records to verify that the append went as expected. First, we can write a query to show the count before we appended new records. Delta Lake can query an earlier version of a Delta table using a feature known as time travel.

We demonstrate querying the data as of version 0, which is the initial conversion of the table from Parquet.

`5 devices * 24 hours * 31 days = 3720 records`

Cmd 18

```
SELECT COUNT(*) FROM health_tracker_silver VERSION AS OF 0
```

▸ (2) Spark Jobs

	count(1)
1	3720

Showing all 1 rows.

Command took 1.79 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:20:26 PM on My Cluster

Count records in our current table

Now, let's count the records to see if our new data was appended as expected. Note that this data is from February 2020, which had 29 days because 2020 was a leap year. We are still working with 5 devices, with heartrate readings occurring once an hour.

`5 devices * 24 hours * 29 days + 3720 = 7200 records`

Cmd 20

```
SELECT COUNT(*) FROM health_tracker_silver
```

▸ (3) Spark Jobs

	count(1)
1	7128

Showing all 1 rows.

Command took 1.81 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:20:41 PM on My Cluster

Find missing records by device

Let's see if we can identify which device(s) are missing records.

The absence of records from the last few days of the month shows a phenomenon that may often occur in a production data pipeline: **late-arriving data**. This can create problems in some of the other data storage and management models we talked about. If our analytics runs on stale or incomplete data, we may draw incorrect conclusions or make bad predicitions. Delta Lake allows us to process data as it arrives and is prepared to handle the occurrence of late arriving data.

Cmd 22

```
1  SELECT p_device_id, COUNT(*) FROM health_tracker_silver GROUP BY p_device_id
```

▸ (3) Spark Jobs

	p_device_id	count(1)
1	0	1440
2	1	1440
3	3	1440
4	2	1440
5	4	1368

Showing all 5 rows.

Command took 2.00 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:20:59 PM on My Cluster

Plot Records

We can run a query and use visualization tools to find out more about which dates or times are missing. For this query, it may be helpful to compare two devices, even though we're showing only one is missing data. Run the cell below to query the table. Then, click on the chart icon to plot the data.

To set up your graph:

- Click `Plot Options`
- Drag `dte` into the Keys dialog
- Drag `p_device_id` into the Series Groupings dialog
- Drag `heartrate` into the values dialog
- Choose `COUNT` as your Aggregation type (the dropdown in the lower left corner)
- Select "Bar Chart" as your display type.

Cmd 24

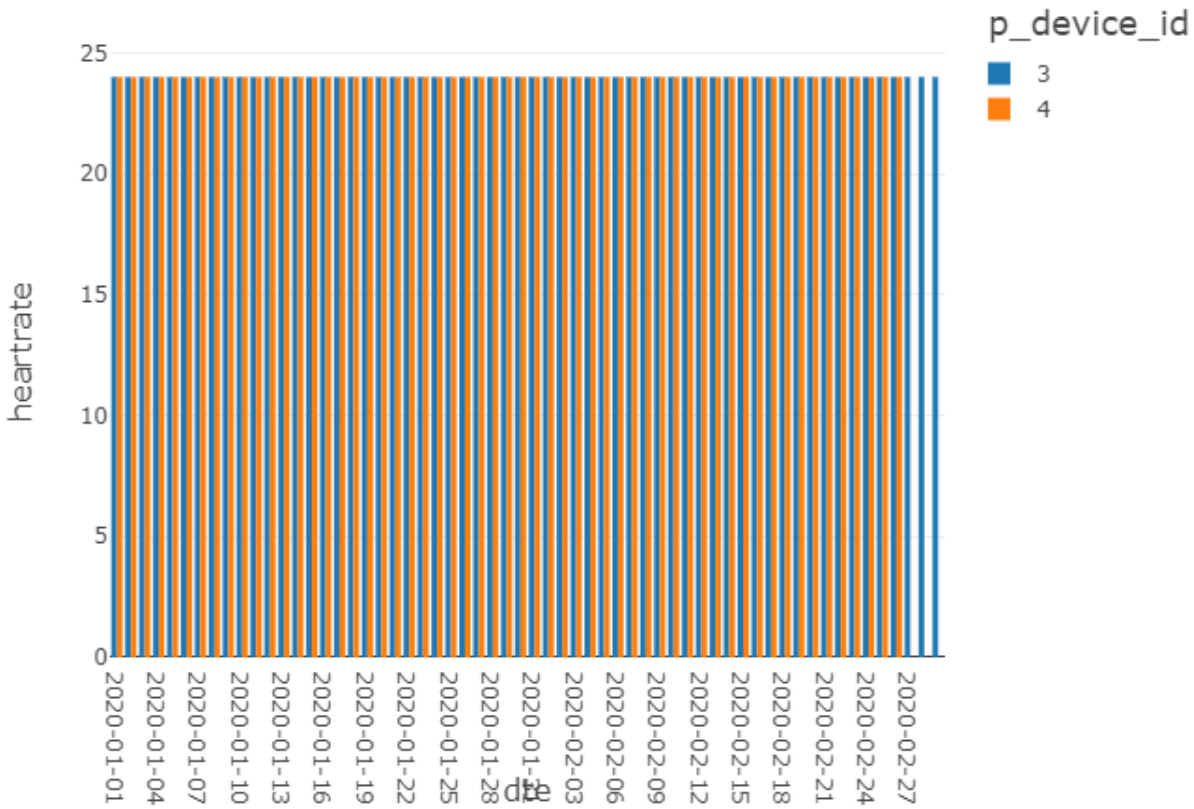
```
1  SELECT * FROM health_tracker_silver WHERE p_device_id IN (3,4)
```

▸ (2) Spark Jobs

	name	heartrate	time	dte	p_device_id
1	Minh Nguyen	54.5276763038	2020-01-01T00:00:00.000+0000	2020-01-01	3
2	Minh Nguyen	55.3566724529	2020-01-01T01:00:00.000+0000	2020-01-01	3
3	Minh Nguyen	55.1554433144	2020-01-01T02:00:00.000+0000	2020-01-01	3
4	Minh Nguyen	56.379849212	2020-01-01T03:00:00.000+0000	2020-01-01	3
5	Minh Nguyen	55.9843632946	2020-01-01T04:00:00.000+0000	2020-01-01	3
6	Minh Nguyen	55.1160133688	2020-01-01T05:00:00.000+0000	2020-01-01	3
7	Minh Nguyen	56.552175579	2020-01-01T06:00:00.000+0000	2020-01-01	3

Truncated results, showing first 1000 rows.

Command took 6.07 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:21:19 PM on My Cluster



Command took 5.91 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:33:44 PM on My Cluster

Find Broken Readings

It's always useful to check for errant readings. Think about this scenario. Is there any reading would seem impossible?

Since this is heartrate date, we should expect that everyone who is using the tracker has a heartbeat. Let's check the data to see if we've got any data that might point to a faulty reading.

We use a temporary view so that we can access this data again later.

Cmd 26

```
CREATE OR REPLACE TEMPORARY VIEW broken_readings
AS (
  SELECT COUNT(*) as broken_readings_count, dte FROM health_tracker_silver
  WHERE heartrate < 0
  GROUP BY dte
  ORDER BY dte
)
```

OK

Command took 0.33 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:21:40 PM on My Cluster

View broken readings

Run the cell and then create a a visualization that will help us get a sense of how many broken readings exist and how they are spread across the data.

To visualize this view:

- Run the cell
- Click the chart icon
- Choose 'dte' as the Key and `broken_readings_count` as Values.

 You should notice that most days have at least one broken reading and that some days have more than one.

Cmd 28

```
1  SELECT * FROM broken_readings;
```

▸ (2) Spark Jobs

	broken_readings_count	dte
1	1	2020-01-01
2	3	2020-01-02
3	3	2020-01-05
4	3	2020-01-06
5	1	2020-01-08
6	1	2020-01-12
7	1	2020-01-15

Showing all 35 rows.

Command took 4.43 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:21:51 PM on My Cluster

Cmd 29

Clean-up

Run the next cell to clean up your classroom enviroment

```
1  %run ../Includes/Classroom-Cleanup
```

Cmd 31

Great job! You're officially working with Delta Lake! In the next reading, you'll continue your work to repair the broken data and missing values we discovered here.

Cmd 32

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

Managing Records

In the previous reading we demonstrated how to create a Delta table. We used basic data exploration strategies to identify two problems within the data. In this notebook, we will demonstrate how to correct those problems and write to a new, clean gold-level table that you can use for queries. Also, we will demonstrate how to repair and correct records

In this notebook, you will:
- Use a window function to interpolate missing values
- Update a Delta table
- Check the version history in a Delta table

Cmd 3

Getting started

Run the cell below to set up your classroom environment

Cmd 4

```
1 %run ../Includes/8-2-Setup
```

```
Command took 31.60 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:38:10 PM on My Cluster
```

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = true
```

```
res2: Boolean = false
```

```
res3: Boolean = false
```

OK

Query returned no results

OK

OK

	num_affected_rows	num_inserted_rows
1	3408	3408

Repairing Records

In the previous reading, we found two problems in the data:

1. We were missing records on a single device for a period of three days
2. There was at least one broken reading (heartrate less than zero) per day in our set

We will start by demonstrating how to merge a set of updates and insertions to repair these problems.

First, we will work on the broken sensor readings. Previously, you used a window function, paired with the `AVG` function, to calculate an average value over a group of rows. Here, we will use a window function, paired with the built-in functions `LAG` and `LEAD`, to interpolate values to replace the broken readings.

`LAG` : fetches data from the previous row. Learn more.
`LEAD` : fetches data from a subsequent row. Learn more.

Examine the code in the next cell.

`line 1` : We create or replace a temporary view names `updates`
`line 2` : We are using a CTAS pattern to create this new view
`line 3` : We select a subgroup of columns to include from the window function defined in lines 5 - 8. Note the expression `(prev_amt+next_amt)/2`. For any missing entry, we calculate a new data point that is the mean of the previous entry and the subsequent entry. These values are defined in the window function below
`line 4` : Indicates the window function as the source. The parenthesis marks the start of the window function
`line 5` : Select all columns from `health_tracker_silver`
`line 6` : `LAG` gets the `heartrate` from the previous row. We define the window by `device_id` and `dte` so that the calculation is applied to each missing value **by device**
`line 9` : Marks the end of the window function
`line 10` : Identifies that this calulation should apply **only** where the heartrate reading is less than 0

Interpolation is a type of estimation where we construct new data points based on a set of known data points.

```
CREATE OR REPLACE TEMPORARY VIEW updates
AS (
  SELECT name, (prev_amt+next_amt)/2 AS heartrate, time, dte, p_device_id
  FROM (
    SELECT *,
    LAG(heartrate) OVER (PARTITION BY p_device_id, dte ORDER BY p_device_id, dte) AS prev_amt,
    LEAD(heartrate) OVER (PARTITION BY p_device_id, dte ORDER BY p_device_id, dte) AS next_amt
    FROM health_tracker_silver
  )
  WHERE heartrate < 0
)
```

OK

Command took 0.47 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:49:38 PM on My Cluster

Cmd 7

Check schema

We will want to use the values in `updates` to update our `health_tracker_silver` table. Let's check both schemas to see if they match.

Cmd 8

```
DESCRIBE updates
```

	col_name	data_type	comment
1	name	string	null
2	heartrate	double	null
3	time	timestamp	null
4	dte	date	null
5	p_device_id	bigint	null

Showing all 5 rows.

Command took 0.10 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:51:13 PM on My Cluster

```
DESCRIBE health_tracker_silver
```

	col_name	data_type	comment
1	name	string	
2	heartrate	double	
3	time	timestamp	
4	dte	date	
5	p_device_id	bigint	
6			
7	# Partitioning		

Showing all 8 rows.

Command took 0.22 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 10:51:19 PM on My Cluster

Cmd 10

Late-arriving data

We're ready to update our silver table with our interpolated values, but before we do, we find out that those missing readings have finally come through! We can get that data ready to merge with our other updates.

Run the cell below to read in the raw data.

Cmd 11

```
DROP TABLE IF EXISTS health_tracker_data_2020_02_late;

CREATE TABLE health_tracker_data_2020_02_late
USING json
OPTIONS (
  path "dbfs:/mnt/training/healthcare/tracker/raw-late.json",
  inferSchema "true"
  );
```

Prepare inserts

We can apply the same transformations we used to create our `health_tracker_silver` table on this raw data. This we'll give us a view with the same schema as our other tables and views.

Cmd 13

```
CREATE OR REPLACE TEMPORARY VIEW inserts AS (
  SELECT
    value.name,
    value.heartrate,
    CAST(FROM_UNIXTIME(value.time) AS timestamp) AS time,
    CAST(FROM_UNIXTIME(value.time) AS DATE) AS dte,
    value.device_id p_device_id
  FROM
    health_tracker_data_2020_02_late
)
```

OK

Command took 0.64 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:01:19 PM on My Cluster

Cmd 14

Prepare upserts

The word "upsert" is a portmanteau of the words "update" and "insert," and this is what it does. An upsert will update records where some criteria are met and otherwise will insert the record. We create a view that is the union of our `updates` and `inserts` and holds all records we would like to add and modify.

Here, we use `UNION ALL` to capture all records in both views, even duplicates.

Cmd 15

```
CREATE OR REPLACE TEMPORARY VIEW upserts
AS (
    SELECT * FROM updates
    UNION ALL
    SELECT * FROM inserts
    )
```

OK

Command took 0.35 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:01:30 PM on My Cluster

Perform upsert

When upserting into an existing Delta table, use Spark SQL to perform the merge from another registered table or view. The Transaction Log records the transaction, and the Metastore immediately reflects the changes.

The merge appends both the new/inserted files and the files containing the updates to the Delta file directory. The transaction log tells the Delta reader which file to use for each record.

This operation is similar to the SQL `MERGE` command but has added support for deletes and other conditions in updates, inserts, and deletes. In other words, using the Spark SQL command `MERGE` provides full support for an upsert operation.

Use the comments to better understand how this command integrates records from our existing tables and views.

Read more about `MERGE INTO` here.

Cmd 17

```
MERGE INTO health_tracker_silver                          -- the MERGE instruction is used to perform the upsert
USING upserts

ON health_tracker_silver.time = upserts.time AND
    health_tracker_silver.p_device_id = upserts.p_device_id  -- ON is used to describe the MERGE condition

WHEN MATCHED THEN                                         -- WHEN MATCHED describes the update behavior
  UPDATE SET
  health_tracker_silver.heartrate = upserts.heartrate
WHEN NOT MATCHED THEN                                     -- WHEN NOT MATCHED describes the insert behavior
  INSERT (name, heartrate, time, dte, p_device_id)
  VALUES (name, heartrate, time, dte, p_device_id)
```

▸ (10) Spark Jobs

	num_affected_rows	num_updated_rows	num_deleted_rows	num_inserted_rows
1	146	74	0	72

Showing all 1 rows.

Command took 21.19 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:04:57 PM on My Cluster

Time travel

Let's check the number of records in the different versions of our tables.

Version 1 shows the data after we added records from February. Recall that this is where we first discovered the missing records.

The current version shows everything including the records we upcerted.

Cmd 19

```
1  -- VERSION 1
2  SELECT COUNT(*) FROM health_tracker_silver VERSION AS OF 1
```

▸ (3) Spark Jobs

	count(1)
1	7128

Showing all 1 rows.

Command took 1.48 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:06:14 PM on My Cluster

Cmd 20

```
1  -- CURRENT VERSION
2  SELECT COUNT(*) FROM health_tracker_silver
```

▸ (3) Spark Jobs

	count(1)
1	7200

Showing all 1 rows.

Command took 0.89 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:06:19 PM on My Cluster

Describe history

You can check the full history of a Delta table including the operation, user, and so on for each new write to a table.

Cmd 22

```
1  DESCRIBE HISTORY health_tracker_silver
```

▸ (1) Spark Jobs

	version	timestamp	userId	userName	operation	operationParameters
1	2	2021-07-08T06:05:16.000+0000	1243378009092075	SWCPROPERTY@GMAIL.COM	MERGE	▸ {"predicate": "((spark_catalog.default.health_tracker_silver.`time` = upserts.`time`) AND (spark_catalog.default.health_tracker_silver.`p_device_id` = upserts.`p_device_id`))", "matchedPredicates": "[{\"actionType\":\"update\"}]", "notMatchedPredicates": "[{\"actionType\":\"insert\"}]"}
2	1	2021-07-08T05:38:39.000+0000	1243378009092075	SWCPROPERTY@GMAIL.COM	WRITE	▸ {"mode": "Append", "partitionBy": "[]"}
3	0	2021-07-08T05:38:27.000+0000	1243378009092075	SWCPROPERTY@GMAIL.COM	CREATE OR REPLACE TABLE AS SELECT	▸ {"isManaged": "false", "description": null, "partitionBy": "[\"p_device_id\"]", "properties": "{}"}

Showing all 3 rows.

Command took 1.81 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:06:56 PM on My Cluster

		notebook	clusterId	readVersion	isolationLevel	isBlindAppend	operationMetrics	userMetadata
1		▸ {"notebookId": "2893064224355773"}	0708-050106-hued619	1	WriteSerializable	false	▸ {"numTargetRowsCopied": "7054", "numTargetRowsDeleted": "0", "numTargetFilesAdded": "5", "executionTimeMs": "16849", "numTargetRowsInserted": "72", "scanTimeMs": "6021", "numTargetRowsUpdated": "74", "numOutputRows": "7200", "numTargetChangeFilesAdded": "0", "numSourceRows": "146", "numTargetFilesRemoved": "10", "rewriteTimeMs": "10720"}	null
2		▸ {"notebookId": "2893064224355773"}	0708-050106-hued619	0	WriteSerializable	true	▸ {"numFiles": "5", "numOutputBytes": "53040", "numOutputRows": "3408"}	null
3		▸ {"notebookId": "2893064224355773"}	0708-050106-hued619	null	WriteSerializable	false	▸ {"numFiles": "5", "numOutputBytes": "57263", "numOutputRows": "3720"}	null

Showing all 3 rows.

Command took 1.81 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:06:56 PM on My Cluster

Write to gold

So far, we have ingested raw (bronze-level) data and applied transformations to create a silver table. We have used Spark SQL to explore and transform that data further, adding new values when we found collection errors and updating the table to reflect late-arriving data. Now that our data is clean and polished, we can write to a gold table. Gold tables are used to hold business level aggregates. When we create this table, we also apply aggregate functions to several columns.

Cmd 24

```
DROP TABLE IF EXISTS health_tracker_gold;

CREATE TABLE health_tracker_gold
USING DELTA
LOCATION "/health_tracker/gold"
AS
SELECT
  AVG(heartrate) AS meanHeartrate,
  STD(heartrate) AS stdHeartrate,
  MAX(heartrate) AS maxHeartrate
FROM health_tracker_silver
GROUP BY p_device_id

```

▸ (5) Spark Jobs

Query returned no results

Command took 6.36 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:07:04 PM on My Cluster

Cmd 25

```
SELECT
  *
FROM
  health_tracker_gold
```

▸ (1) Spark Jobs

	meanHeartrate	stdHeartrate	maxHeartrate
1	87.56374275065477	28.697966782294422	191.7364805027
2	82.8938339413805	24.627453529979476	174.3611679317
3	82.49632786583678	22.460895905499	192.1828472326
4	86.40550075283315	25.919716960302623	177.6570899987
5	80.99397568360457	23.832859676185365	175.8194964652

Showing all 5 rows.

Command took 0.70 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:07:22 PM on My Cluster

Cmd 26

Cleanup

Run the next cell to clean up your classroom environment

Cmd 27

```
%run ../Includes/Classroom-Cleanup
```

Command took 0.04 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:09:12 PM on My Cluster

Cmd 28

Great work! Now that you've got a basic understanding of how data moves through the Delta architecture, we're ready to get back to analytics. In the next reading, we'll see how to write high-performance Spark queries with Databricks Delta.

Cmd 29

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

8.3 Optimizing Delta (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear

Cmd 2

Optimizing Delta

In this notebook, you'll see some examples of how you can optimize your queries using Delta Engine, which is built-in to the Databricks Runtime 7.0. It is also part of open source Delta Lake.

The data contains information about US-based flight schedules from 2008. It is made available to us via Databricks Datasets.

First, we will create a standard table using Parquet format and then we'll run a query to observe the timing.

Then, we'll run the same query on a Delta table using Delta Engine optimizations and compare the two.

Databricks includes a variety of datasets that you can use to continue learning, or just for practice! Check out the docs for copyable Python code that you can use to see what sets are available.

Run the cell below to set up your classroom environment.

Cmd 3

```
%run ../Includes/Classroom-Setup
```

Command took 7.39 seconds -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:29:29 PM on My Cluster

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = true

res2: Boolean = true
```

Cmd 4

Create a Parquet table

Run the command below to create a Parquet table.

```sql
DROP TABLE IF EXISTS flights;
-- Create a standard table and import US based flights for year 2008
-- USING Clause: Specify parquet format for a standard table
-- PARTITIONED BY clause: Orginize data based on "Origin" column (Originating Airport code).
-- FROM Clause: Import data from a csv file.
CREATE TABLE flights
USING
  parquet
PARTITIONED BY
  (Origin)
SELECT
  _c0 AS Year,
  _c1 AS MONTH,
  _c2 AS DayofMonth,
  _c3 AS DayOfWeek,
  _c4 AS DepartureTime,
  _c5 AS CRSDepartureTime,
  _c6 AS ArrivalTime,
  _c7 AS CRSArrivalTime,
  _c8 AS UniqueCarrier,
  _c9 AS FlightNumber,
  _c10 AS TailNumber,
  _c11 AS ActualElapsedTime,
  _c12 AS CRSElapsedTime,
  _c13 AS AirTime,
  _c14 AS ArrivalDelay,
  _c15 AS DepartureDelay,
  _c16 AS Origin,
  _c17 AS Destination,
  _c18 AS Distance,
  _c19 AS TaxiIn,
  _c20 AS TaxiOut,
  _c21 AS Cancelled,
  _c22 AS CancellationCode,
  _c23 AS Diverted,
  _c24 AS CarrierDelay,
  _c25 AS WeatherDelay,
  _c26 AS NASDelay,
  _c27 AS SecurityDelay,
  _c28 AS LateAircraftDelay
FROM                                -- This table is being read in directly from a csv file.
  csv.`dbfs:/databricks-datasets/asa/airlines/2008.csv`
```

Command took 5.83 minutes -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:29:55 PM on My Cluster

Cmd 6

Highest monthly total (Parquet)

Run the query to get the top 20 cities with the highest monthly total flights on the first day of the week. Be sure to note the time when the query finishes.

Cmd 7

```
SELECT Month, Origin, count(*) as TotalFlights
FROM flights
WHERE DayOfWeek = 1
GROUP BY Month, Origin
ORDER BY TotalFlights DESC
LIMIT 20;
```

▸ (3) Spark Jobs

	Month	Origin	TotalFlights
1	6	ATL	6046
2	3	ATL	6019
3	12	ATL	5800
4	9	ATL	5722
5	6	ORD	5241
6	3	ORD	5072
7	9	ORD	4931

Showing all 20 rows.

Command took 3.75 minutes -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:39:02 PM on My Cluster

Cmd 8

Create a Delta Table

Run the query below to compare Delta to Parquet. Note, this is the exact same command running on the exact same cluster configuration. Recall that the two operations take roughly the same amount of "work" from Spark. We have to read in a huge csv file, partition it by origin, and store it in a new, columnar format. Plus, Delta is creating a transaction log and tagging the files with important and useful metadata!

```sql
DROP TABLE IF EXISTS flights;
-- Create a standard table and import US based flights for year 2008
-- USING Clause: Specify "delta" format instead of the standard parquet format
-- PARTITIONED BY clause: Orginize data based on "Origin" column (Originating Airport code).
-- FROM Clause: Import data from a csv file.
CREATE TABLE flights
USING
  delta
PARTITIONED BY
  (Origin)
SELECT
  _c0 AS Year,
  _c1 AS MONTH,
  _c2 AS DayofMonth,
  _c3 AS DayOfWeek,
  _c4 AS DepartureTime,
  _c5 AS CRSDepartureTime,
  _c6 AS ArrivalTime,
  _c7 AS CRSArrivalTime,
  _c8 AS UniqueCarrier,
  _c9 AS FlightNumber,
  _c10 AS TailNumber,
  _c11 AS ActualElapsedTime,
  _c12 AS CRSElapsedTime,
  _c13 AS AirTime,
  _c14 AS ArrivalDelay,
  _c15 AS DepartureDelay,
  _c16 AS Origin,
  _c17 AS Destination,
  _c18 AS Distance,
  _c19 AS TaxiIn,
  _c20 AS TaxiOut,
  _c21 AS Cancelled,
  _c22 AS CancellationCode,
  _c23 AS Diverted,
  _c24 AS CarrierDelay,
  _c25 AS WeatherDelay,
  _c26 AS NASDelay,
  _c27 AS SecurityDelay,
  _c28 AS LateAircraftDelay
FROM
  csv.`dbfs:/databricks-datasets/asa/airlines/2008.csv`;
```

Command took 6.39 minutes -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:43:02 PM on My Cluster

Cmd 10

Optimize your table

If your organization continuously writes data to a Delta table, it will over time accumulate a large number of files, especially if you add data in small batches. For analysts, a common complaint in querying data lakes is read efficiency; and having a large collection of small files to sift through everytime data is queried can create performance problems. Ideally, a large number of small files should be rewritten into a smaller number of larger files on a regular basis, which will improve the speed of read queries from a table. This is known as compaction. You can compact a table using the `OPTIMIZE` command shown below.

Z-ordering co-locates column information (recall that Delta is columnar storage). Co-locality is used by Delta Lake data-skipping algorithms to dramatically reduce the amount of data that needs to be read. You can specify multiple columns for ZORDER BY as a comma-separated list. However, the effectiveness of the locality drops with each additional column. Read more about optimizing Delta tables here.

Cmd 11

```
1 OPTIMIZE flights ZORDER BY (DayofWeek);
```

▸ (4) Spark Jobs

	path	metrics
1	null	▸ {"numFilesAdded": 300, "numFilesRemoved": 2308, "filesAdded": {"min": 8696, "max": 7000960, "avg": 384121.41333333333, "totalFiles": 300, "totalSize": 115236424}, "filesRemoved": {"min": 6376, "max": 1141391, "avg": 62792.94887348353, "totalFiles": 2308, "totalSize": 144926126}, "partitionsOptimized": 304, "zOrderStats": {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputOtherFiles": {"num": 2312, "size": 144955254}, "inputNumCubes": 0, "mergedFiles": {"num": 2308, "size": 144926126}, "numOutputCubes": 300, "mergedNumCubes": null}, "numBatches": 1, "totalConsideredFiles": 2312, "totalFilesSkipped": 4, "preserveInsertionOrder": false}

Showing all 1 rows.

Command took 13.25 minutes -- by SWCPROPERTY@GMAIL.COM at 7/7/2021, 11:49:43 PM on My Cluster

Cmd 12

Rerun the query

Run the query below to compare performance for a standard Parquet table with an optimized Delta table.

```
1 SELECT Month, Origin, count(*) as TotalFlights
2 FROM flights
3 WHERE DayOfWeek = 1
4 GROUP BY Month, Origin
5 ORDER BY TotalFlights DESC
6 LIMIT 20;
```

▸ (2) Spark Jobs

	Month	Origin	TotalFlights
1	6	ATL	6046
2	3	ATL	6019
3	12	ATL	5800
4	9	ATL	5722
5	6	ORD	5241
6	3	ORD	5072
7	9	ORD	4931

Showing all 20 rows.

Command took 33.53 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 12:03:32 AM on My Cluster

Cmd 14

Delta Cache

Using the Delta cache is an excellent way to optimize performance. Note: The Delta cache is *not* the same as caching in Apache Spark, which we talked about in Module 4. One notable difference is that the Delta cache is stored entirely on the local disk, so that memory is not taken away from other operations within Spark. When enabled, the Delta cache automatically creates a copy of a remote file in local storage so that successive reads are significantly sped up. Unfortunately, to enable it, you must choose a cluster type that is not available in Databricks Community Edition.

To better understand the differences between Delta caching and Apache Spark caching, please read, "Delta and Apache Spark caching."

Cmd 15

```
1 %run ../Includes/Classroom-Cleanup
2
```

Command took 0.04 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 12:04:17 AM on My Cluster

8.4 Lab - Delta Lab (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revision hi

Lab 4 - Delta Lab

Module 8 Assignment

In this lab, you will continue your work on behalf of Moovio, the fitness tracker company. You will be working with a new set of files that you must move into a "gold-level" table. You will need to modify and repair records, create new columns, and merge late-arriving data.

Cmd 3

```
1 %run ../Includes/Classroom-Setup
```

Command took 4.43 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 9:42:54 PM on My Cluster

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res4: Boolean = true

res5: Boolean = false
```

Cmd 4

Exercise 1: Create a table

Summary: Create a table from `json` files.

Use this path to access the data:
`"dbfs:/mnt/training/healthcare/tracker/raw.json/"`

Steps to complete:
- Create a table named `health_tracker_data_2020`
- Use optional fields to indicate the path you're reading from and epress that the schema should be inferred.

```
1  Drop table if exists health_tracker_data_2020;
2  Create table health_tracker_data_2020
3  Using json
4  Options (
5    path "dbfs:/mnt/training/healthcare/tracker/raw.json/",
6    inferschema "true"
7  )
```

▸ (1) Spark Jobs

OK

Command took 2.96 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 9:43:07 PM on My Cluster

Cmd 6

Exercise 2: Preview the data

Summary: View a sample of the data in the table.

Steps to complete:

- Query the table with `SELECT *` to see all columns
- Sample 5 rows from the table

Cmd 7

```
1  select * from health_tracker_data_2020 Tablesample (5 rows)
```

▸ (1) Spark Jobs

	month	value
1	2020-05	▸ {"device_id": 0, "heartrate": 54.7922842229, "name": "Deborah Powell", "time": 1588291200}
2	2020-05	▸ {"device_id": 0, "heartrate": 56.1916535912, "name": "Deborah Powell", "time": 1588294800}
3	2020-05	▸ {"device_id": 0, "heartrate": 56.491746118, "name": "Deborah Powell", "time": 1588298400}
4	2020-05	▸ {"device_id": 0, "heartrate": 55.9563823115, "name": "Deborah Powell", "time": 1588302000}
5	2020-05	▸ {"device_id": 0, "heartrate": 56.1483078922, "name": "Deborah Powell", "time": 1588305600}

Showing all 5 rows.

Command took 0.92 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 9:43:15 PM on My Cluster

Exercise 3: Count Records

Summary: Write a query to find the total number of records

Steps to complete:

- Count the number of records in the table

Answer the corresponding question in Coursera

Cmd 9

```
1  select count(*) from health_tracker_data_2020
```

▸ (2) Spark Jobs

	count(1)
1	18168

Showing all 1 rows.

Command took 0.97 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 9:43:27 PM on My Cluster

Cmd 10

Exercise 4: Create a Silver Delta table

Summary: Create a Delta table that transforms and restructures your table

Steps to complete:

- Drop the existing `month` column
- Isolate each property of the object in the `value` column to its own column
- Cast time as timestamp **and** as a date
- Partition by `device_id`
- Use Delta to write the table

```
Create or Replace Table health_tracker_data_2020_silver
Using Delta
Partitioned by (p_device_id)
--location "/health_tracker/silver"
 as (
Select
  value.heartrate,
  value.name,
  Cast(From_Unixtime(value.time) as timestamp) as time,
  Cast(From_Unixtime(value.time) as date) as dte,
  value.device_id p_device_id
From
  health_tracker_data_2020
)
```

▸ (4) Spark Jobs

Query returned no results

Command took 12.67 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 9:44:01 PM on My Cluster

Cmd 12

Exercise 5: Register table to the metastore

Summary: Register your Silver table to the Metastore Steps to complete:

- Be sure you can run the cell more than once without throwing an error
- Write to the location: `/health_tracker/silver`

Cmd 13

```
Describe detail health_tracker_data_2020_silver
```

	format	id	name	description	location	createdAt	lastModified	partitionColumns	numl
1	delta	094291a0-5cdb-448b-a676-fd7305cc8209	default.health_tracker_data_2020_silver	null	dbfs:/user/hive/warehouse/health_tracker/silver	2021-07-09T04:44:02.458+0000	2021-07-09T04:44:09.000+0000	▸ ["p_device_id"]	25

Showing all 1 rows.

Command took 0.49 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 9:44:24 PM on My Cluster

Exercise 6: Check the number of records

Summary: Check to see if all devices are reporting the same number of records

Steps to complete:

- Write a query that counts the number of records for each device
- Include your partitioned device id column and the count of those records

Answer the corresponding question in Coursera

Cmd 15

```
1  select p_device_id, count(*)
2  from health_tracker_data_2020_silver
3  group by p_device_id
```

▸ (2) Spark Jobs

	p_device_id	count(1)
1	0	3648
2	1	3648
3	2	3648
4	3	3648
5	4	3576

Showing all 5 rows.

Command took 3.06 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 9:47:53 PM on My Cluster

Exercise 7: Plot records

Summary: Attempt to visually assess which dates may be missing records

Steps to complete:

- Write a query that will return records from one devices that is **not** missing records as well as the device that seems to be missing records
- Plot the results to visually inspect the data
- Identify dates that are missing records

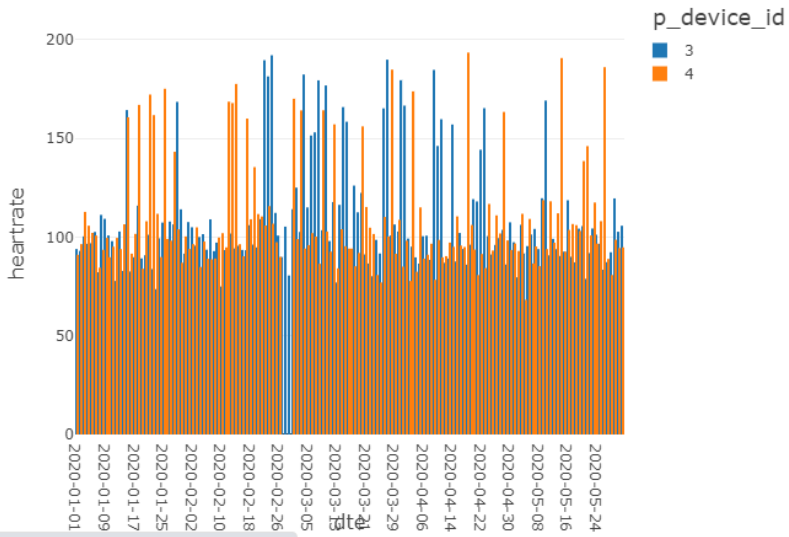
Answer the corresponding question in Coursera

Cmd 17

```
1  select * from health_tracker_data_2020_silver where p_device_id IN (3,4)
```

▸ (9) Spark Jobs

Aggregated (by max) in the backend.



Exercise 8: Check for Broken Readings

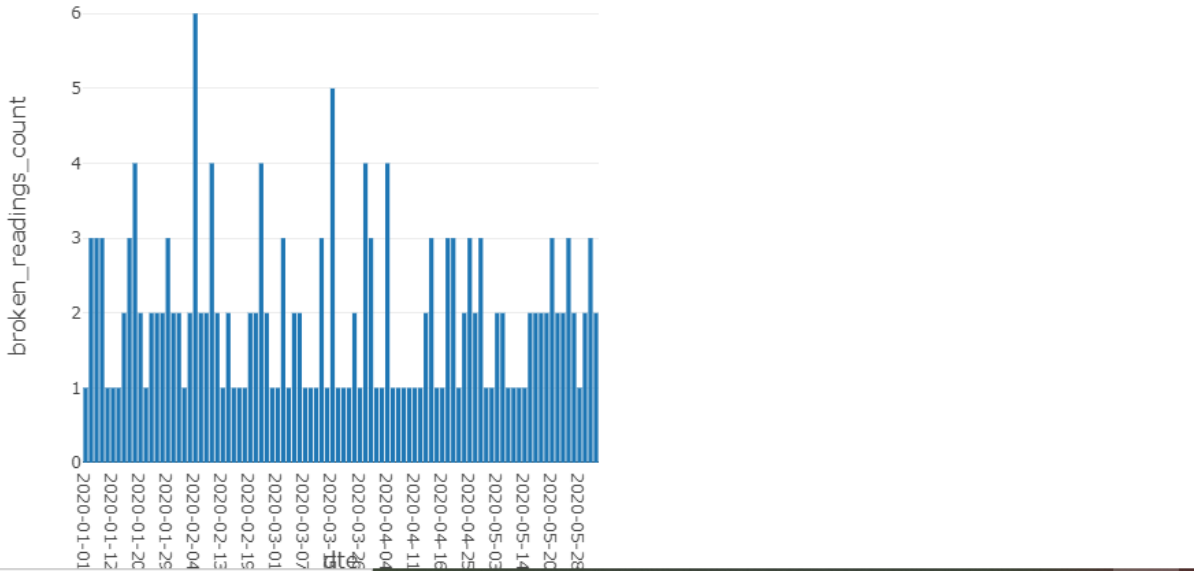
Summary: Check to see if your data contains records that would indicate a device has misreported data Steps to complete:

- Create a view that contains all records reporting a negative heartrate
- Plot/view that data to see which days include broken readings

Cmd 19

```
1  Create or Replace Temporary View Broken_Readings
2  AS (
3    Select dte, count(*) as broken_readings_count
4    from health_tracker_data_2020_silver
5    where heartrate < 0
6    group by dte
7    order by dte
8  );
9  select * from Broken_Readings;
```

▸ (2) Spark Jobs



Exercise 9: Repair records

Summary: Create a view that contains interpolated values for broken readings

Steps to complete:

- Create a temporary view that will hold all the records you want to update.
- Transform the data such that all broken readings (where heartrate is reported as less than zero) are interpolated as the mean of the the data points immediately surrounding the broken reading.
- After you write the view, count the number of records in it.

Answer the corresponding question in Coursera

Cmd 21

```
Create or Replace Temporary View updates
AS (
  Select
    (prev_amt+next_amt)/2 as heartrate,
    name, time, dte, p_device_id
  From (
     Select *,
       LAG(heartrate) over (partition by p_device_id, dte order by p_device_id, dte) As prev_amt,
       LEAD(heartrate) over (partition by p_device_id, dte order by p_device_id, dte) As next_amt
     From
       health_tracker_data_2020_silver
   )
  Where heartrate < 0
  );

Select count(*) from updates;
```

▸ (2) Spark Jobs

	count(1)
1	182

Showing all 1 rows.

Command took 3.86 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:04:36 PM on My Cluster

Exercise 10: Read late-arriving data

Summary: Read in new late-arriving data

Steps to complete:

- Create a new table that contains the late arriving data at this path: `"dbfs:/mnt/training/healthcare/tracker/raw-late.json"`
- Count the records

Answer the corresponding question in Coursera

Cmd 23

```
Drop Table If Exists health_tracker_data_2020_late;
Create table health_tracker_data_2020_late
using json
options (
  path "dbfs:/mnt/training/healthcare/tracker/raw-late.json",
  inferschema "true"
);
select count(*) from health_tracker_data_2020_late;
```

▸ (3) Spark Jobs

	count(1)
1	72

Showing all 1 rows.

Command took 1.94 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:07:02 PM on My Cluster

Exercise 11: Prepare inserts

Summary: Prepare your new, late-arriving data for insertion into the Silver table

Steps to complete:

- Create a temporary view that holds the new late-arriving data
- Apply transformations to the data so that the schema matches our existing Silver table

Cmd 25

```
1  Create or Replace Temporary View inserts as (
2    select
3      value.heartrate,
4      value.name,
5      cast(from_unixtime(value.time) as timestamp) as time,
6      cast(from_unixtime(value.time) as date) as dte,
7      value.device_id p_device_id
8    from
9      health_tracker_data_2020_late
10 )
```

OK

```
Command took 0.06 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:10:26 PM on My Cluster
```

Exercise 12: Prepare upserts

Summary: Prepare a view to upsert to our Silver table

Steps to complete:

- Create a temporary view that is the `UNION` of the views that hold data you want to insert and data you want to update
- Count the records

Answer the corresponding question in Coursera

Cmd 27

```
1  Create or Replace Temporary View upserts as (
2    select * from updates
3    UNION ALL
4    select * from inserts
5  );
6  select count(*) from upserts;
```

▸ (2) Spark Jobs

	count(1)
1	254

Showing all 1 rows.

Command took 4.40 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:11:47 PM on My Cluster

Exercise 13: Perform upserts

Summary: Merge the upserts into your Silver table

Steps to complete:

- Merge data on the time and device id columns from your Silver table and your upserts table
- Use `MATCH` conditions to decide whether to apply an update or an insert

Cmd 29

```
1  Merge into health_tracker_data_2020_silver
2  using upserts
3
4  on health_tracker_data_2020_silver.time = upserts.time AND
5     health_tracker_data_2020_silver.p_device_id = upserts.p_device_id
6
7  when matched then
8    Update set
9    health_tracker_data_2020_silver.heartrate = upserts.heartrate
10 when not matched then
11   Insert (heartrate, name, time, dte, p_device_id)
12   values (heartrate, name, time, dte, p_device_id)
```

▸ (10) Spark Jobs

	num_affected_rows	num_updated_rows	num_deleted_rows	num_inserted_rows
1	254	182	0	72

Showing all 1 rows.

Command took 28.88 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:18:37 PM on My Cluster

Exercise 14: Write to gold

Summary: Create a Gold level table that holds aggregated data

Steps to complete:

- Create a Gold-level Delta table
- Aggregate heartrate to display the average and standard deviation for each device.
- Count the number of records

Cmd 31

```
Drop Table If Exists health_tracker_data_2020_gold;
Create Table health_tracker_data_2020_gold
Using Delta
--Location "/health_tracker/gold"
As
  Select avg(heartrate) as avgHeartRate,
         std(heartrate) as stdHeartRate,
         count(*) as numberOfRecords
  from
    health_tracker_data_2020_silver
  group by p_device_id
;
select * from health_tracker_data_2020_gold;
```

▸ (6) Spark Jobs

	avgHeartRate	stdHeartRate	numberOfRecords
1	82.52207869094194	23.375608427849777	3648
2	85.08801733820111	27.41188410264521	3648
3	84.19046229191886	24.61892380223707	3648
4	84.5435245360994	25.57932106284926	3648
5	82.7775300853638	25.54242733866919	3648

Showing all 5 rows.

Command took 6.80 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:25:42 PM on My Cluster

01 - Basic Queries (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear

Create Tables

Run the cell below to create tables for the questions in this notebook.

Cmd 3

```
1  %run ../Utilities/01-CreateTables
```

```
Command took 27.30 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:37:21 PM on My Cluster
```

Declared the following table:
- **discounts**

Declared the following table:
- **discounts2**

Cmd 4

Question 1: Modify a Table

Summary

Modify the columns in table `discounts` to match the provided schema.

Steps to Complete

Write a SQL query that achieves the following:
- Selects columns `discountId`, `code`, and `price`
- Converts column `discountId` to type `Long`
- Converts column `price` to type `Double`, multiplies it by 100 and then converts to type `Integer`
- Saves this to a temporary view named `q1Results`

```sql
-- TODO Answer 1
Create or Replace Temporary View q1Results as (
  Select
    cast(discountId as long) as discountId,
    code,
    cast(cast(price as double) * 100 as integer) as price
  From discounts
);
select * from q1Results limit 5;
```

▸ (1) Spark Jobs

	discountId	code	price
1	11410081877875674	nPTOQUbY4XT0	689634
2	672107589035079276	0IGcFHWlrO3q	76191
3	5785237762507825515	XZaBeG9iLgIN	928292
4	2426294815284887341	WjKMpEkk56ZU	38747
5	4355249653680192916	Et19MC9mnOM1	552526

Showing all 5 rows.

Command took 1.77 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:44:27 PM on My Cluster

Question 2: Basic Math and Drop Columns

Summary

Modify the columns in table `discounts2` to match the provided schema.

Steps to complete

Write a SQL query on the table `discounts2` that achieves the following:

- Converts column `active` to type `Boolean`
- Creates the column `price` by converting the column `cents` to type `Double` and dividing by 100
- Drops the `cents` column
- Saves this to a temporary view named `q2Results`

Cmd 7

```
-- TODO Answer 2
Create or Replace Temporary View q2Results as (
  Select
    cast(active as boolean) as active,
    (cast(cents as double) / 100) as price
  From
    discounts2
);
select * from q2Results limit 5;
```

▸ (1) Spark Jobs

	active	price
1	false	6896.34
2	true	761.91
3	true	9282.92
4	true	387.47
5	true	5525.26

02 - Manage Query Results (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All

Create Tables

Run the cell below to create tables for the questions in this notebook.

Cmd 3

```
1  %run ../Utilities/02-CreateTables
```

(4) Spark Jobs

Command took 15.93 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 10:49:57 PM on My Cluster

Question 1: Dedupe Sort

Summary

Deduplicate and sort data on products that need to be restocked in a grocery store.

Steps to complete

Write a SQL query on the `products` table that achieves the following:

- Removes duplicate rows
- Sorts rows by the `aisle` column in ascending order (with nulls appearing last), and then by `price` in ascending order
- Stores the results into a temporary view named `q1Results`

A properly completed solution should return both a DataFrame that looks similar to this:

itemId	amount	aisle	price
9958	64	2	2
1432	23	3	24
3242	14	5	5
...	...	...	...
7064	34	null	24
0244	7	null	36

```sql
-- TODO   Answer 1
Create or Replace Temporary View q1Results as (
  Select
    itemId,
    amount,
    aisle,
    price
  from
    products
  group by 1,2,3,4
  order by aisle asc nulls last, price asc
);
select * from q1Results;
```

▸ (2) Spark Jobs

	itemId	amount	aisle	price
1	9958	64	2	2
2	1432	23	3	24
3	3242	14	5	5
4	9382	15	7	8
5	7012	129	10	11
6	6693	17	14	3
7	6914	5	null	11

Showing all 9 rows.

Command took 0.95 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 11:26:58 PM on My Cluster

Question 2: Limit Results

Summary

Return the top five results for data that matches a set of criteria.

Steps to complete

Write a SQL query on the table `raceResults` that achieves the following:

- Casts `lastFinish` column as date and renames to `raceDate`
- Sorts rows by the `winOdds` column in descending order
- Limits the results to the top 5 `winOdds`
- Stores the results into a temporary view named `q2Results`

A properly completed solution should return a DataFrame that looks similar to this:

name	winOdds	raceDate
Dolor Incididunt	.9634252	2015-07-29
Excepteur Mollit	.9524401	2019-08-15
Magna Ad	.9420442	2017-05-12
Sed In	.9325211	2014-08-29
Qui Cupidat	.9242451	2011-08-23

```sql
--TODO  Answer 2
create or replace temporary view q2Results as (
  Select
    name,
    winOdds,
    to_date(lastFinish) as raceDate
  From
    raceResults
  Order by winOdds desc
);
Select * from q2Results limit 5;
```

▸ (1) Spark Jobs

	name	winOdds	raceDate
1	Esse Tempor	0.99982446	2000-02-21
2	Minim	0.99977535	2021-02-06
3	Adipiscing Eu	0.999763	2009-05-03
4	Velit Do	0.9996125	2009-02-01
5	Excepteur In	0.999441	2004-04-20

Showing all 5 rows.

Command took 1.59 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 11:32:37 PM on My Cluster

Cmd 8

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

03 - Joins (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All

Create Tables

Run the cell below to create tables for the questions in this notebook.

Cmd 3

```
1  %run ../Utilities/03-CreateTables
```

(8) Spark Jobs

Command took 22.10 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 11:36:11 PM on My Cluster

Cmd 4

Question 1: Joins

Summary

Perform a join of two tables `purchases` and `prices`.

Steps to complete

Write a SQL query that acheives the following:

- Performs an inner join on two tables `purchases` and `prices` on the column `itemId`
- Includes **only** 3 columns from the `purchases` table: `transactionId`, `itemId`, and `value`

Cmd 5

```
1  --TODO  Answer 1
2  select transactionId, purchases.itemId, value
3  from purchases
4    inner join prices
5    on purchases.itemId = prices.itemId
```

(1) Spark Jobs

	transactionId	itemId	value
1	3072	721798132696267935	7639.71
2	3073	5274937845866294368	593.41
3	3074	1184079848143569840	495.42

Question 2: Combine Tables

Summary

Perform an outer join on two tables `discounts` and `products` store the results into `q2Results` view.

Steps to complete

Write a SQL query that achieves the following:

- Performs an outer join on table `discounts` and `products` on the `itemName` column
- Ensures that the joined view only includes **one** `itemName` column that comes from the `products` table

The final schema and DataFrame should contain the following columns, though not necessarily in this order:

column	type
itemName	string
discountId	integer
discountCode	string
price	double
active	boolean
itemId	integer
amount	integer

Cmd 7

```
-- TODO  Answer 2
Create or Replace Temporary View q2Results as (
  select products.itemName, discountId, discountcode, price, active, itemId, amount
  from products
    left outer join discounts
    on products.itemName = discounts.itemName
);
select * from q2Results limit 5;
```

	itemName	discountId	discountcode	price	active	itemId	amount
1	Aliqua Dolore	182464	TKA8zpq7ylfH	3.52	false	26631	2199
2	Aliqua Ea	null	null	null	null	67697	323
3	Aliqua Id	null	null	null	null	83723	362
4	Aliqua Minim	968504	ql2a5pyV12PecVX2tFE	64.6	true	21837	190
5	Excepteur Enim	217000	PlrHip77dC1D	23.73	false	29337	325

Showing all 5 rows.

Command took 1.31 seconds -- by SWCPROPERTY@GMAIL.COM at 7/8/2021, 11:59:33 PM on My Cluster

Cmd 8

Question 3: Cross Join Tables

Summary

Perform a cross join on two tables `stores` and `articles`.

Steps to complete

- Perform a cross join on two tables `stores` and `articles`

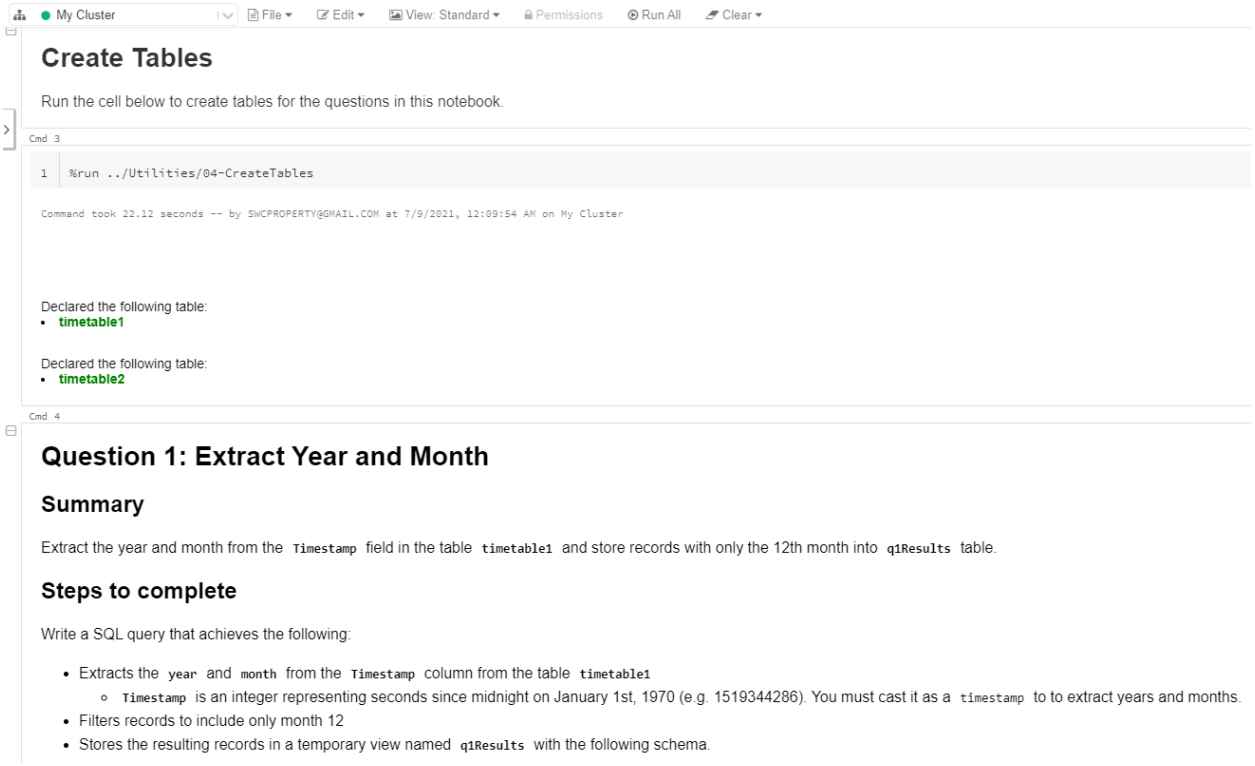
Cmd 9

```
1  -- TODO  Answer 3
2  select * from stores cross join articles
```

(4) Spark Jobs

	storeName	storeId	itemName	itemId	amount
1	Excepteur Aute	172	Pariatur Incididunt	76589	473
2	Excepteur Aute	172	Aliqua Ea	67697	323
3	Excepteur Aute	172	Aliqua Minim	21837	190
4	Excepteur Aute	172	Excepteur Officia	12881	89
5	Excepteur Aute	172	Aliqua Dolore	26631	2199

04 - Timestamp Functions (SQL)



column	type
Date	timestamp
Year	integer
Month	integer

A properly completed solution should produce a DataFrame similar to this sample output:

Date	Year	Month
2010-12-15 21:36:55	2010	12
2002-12-01 11:17:54	2002	12
2017-12-13 11:28:03	2017	12

Cmd 5

```
-- TODO  Answer 1
Drop table if exists q1Results;
Create table q1Results (
Select
  cast(from_unixtime(Timestamp) as timestamp) as Date,
  Year(cast(timestamp as timestamp)) as Year,
  Month(cast(timestamp as timestamp)) as Month
From
  timetable1
Where
  Month(cast(timestamp as timestamp)) = 12
);
select * from q1Results;
```

▸ (7) Spark Jobs

	Date	Year	Month
1	2006-12-16T00:51:29.000+0000	2006	12
2	2014-12-16T08:10:54.000+0000	2014	12
3	2005-12-31T23:11:53.000+0000	2005	12

Question 2: Extract year, month, and day of year

Summary

Extract the **year**, **month** and **dayofyear** from the `Timestamp` field in the table `timetable2` and return records for **only** the 4th month.

Steps to complete

Write a SQL query that achieves the following:

- Create `Year` , `Month` and `DayOfYear` columns from the `Timestamp` column in the `timetable2` table
  - `Timestamp` is an integer representing seconds since midnight on January 1st, 1970 (e.g. 1519344286). You must cast it as a `timestamp` to to extract years, months, and day of year.
- Filter records to include only month 4
- Stores the records in a temporary table named `q2Results` with the following schema:

column	type
Date	timestamp
Year	integer
Month	integer
DayOfYear	integer

- A properly completed solution should produce a DataFrame similar to this sample output:

Date	Year	Month	DayOfYear
2002-04-22 06:41:39	2002	4	112
2012-04-01 05:00:06	2012	4	92
2019-04-05 12:38:42	2019	4	95

```sql
-- TODO  Answer 2
Create or Replace Temporary View q2Results as (
  select
    cast(from_unixtime(Timestamp) as timestamp) as Date,
    Year(cast(timestamp as timestamp)) as Year,
    Month(cast(timestamp as timestamp)) as Month,
    Date_format(cast(timestamp as timestamp), "D") as DayOfYear
  from
    timetable2
  where
    Month(cast(timestamp as timestamp)) = 4
);
select * from q2Results;
```

▸ (1) Spark Jobs

	Date	Year	Month	DayOfYear
1	2000-04-26T07:39:56.000+0000	2000	4	117
2	2020-04-28T12:39:59.000+0000	2020	4	119
3	2017-04-08T20:17:42.000+0000	2017	4	98
4	2002-04-24T17:49:49.000+0000	2002	4	114
5	2013-04-12T05:52:00.000+0000	2013	4	102
6	2017-04-03T21:33:38.000+0000	2017	4	93
7	2005-04-24T16:01:10.000+0000	2005	4	114

Showing all 889 rows.

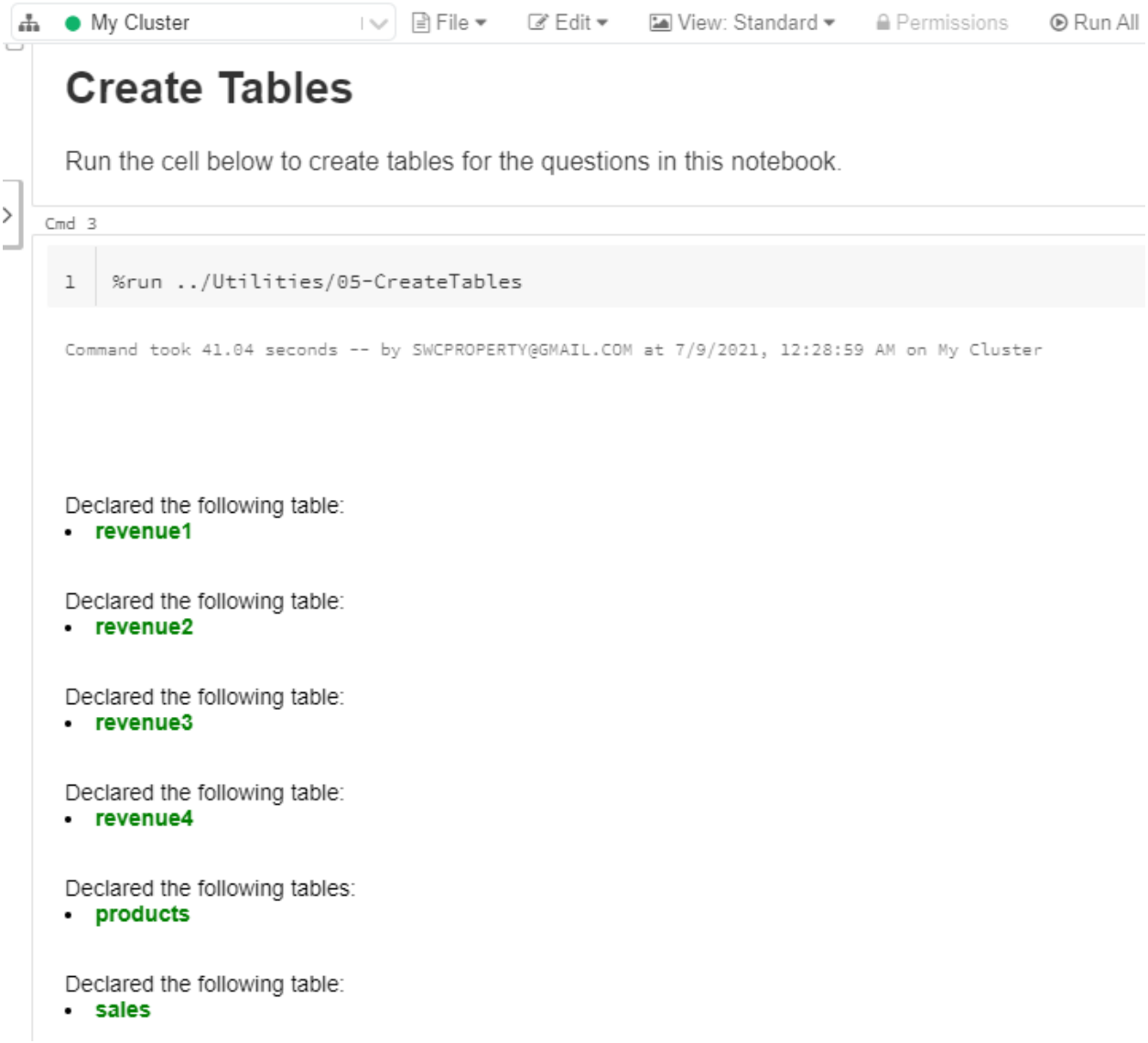
Command took 1.23 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 12:26:15 AM on My Cluster

Cmd 8

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

Privacy Policy | Terms of Use | Support

05 - Aggregate Functions (SQL)



Question 1: Min Function

Summary

Compute the minimum value from the `Amount` field for each unique value in the `TrueFalse` field in the table `revenue1`.

Steps to complete

Write a SQL query that achieves the following:

- Computes the number of `true` and `false` records in the `TrueFalse` field from the table `revenue1`
- Renames the new column to `count`
- Store the records in a temporary view named `q1Results` with the following schema:

column	type
TrueFalse	boolean
MinAmount	int

A properly completed solution should produce a view similar to this sample output:

TrueFalse	count
true	4956
false	5044

Cmd 5

```
-- TODO  Answer 1
Create or Replace Temporary View q1Results as (
  Select TrueFalse, count(*) as count
  from revenue1
  group by TrueFalse
);
select * from q1Results;
```

	TrueFalse	count
1	true	4967
2	false	5033

Showing all 2 rows.

Command took 2.23 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 12:31:49 AM on My Cluster

Cmd 6

Question 2: Max Function

Summary

Compute the maximum value from the `Amount` field for each unique value in the `TrueFalse` field in the table `revenue2`.

Steps to complete

- Computes the maximum `Amount` for `True` records and `False` records from the `TrueFalse` field from the table `revenue2`
- Renames the new column to `maxAmount`
- Store the records in a temporary view named `q2Results` with the following schema:

column	type
TrueFalse	boolean
maxAmount	double

A properly completed solution should produce a DataFrame similar to this sample output:

TrueFalse	MaxAmount
true	2243937.93
false	2559457.1799999997

```sql
-- TODO  Answer 2
Create or Replace Temporary View q2Results as (
  select TrueFalse, max(Amount) as maxAmount
  from revenue2
  group by TrueFalse
);
select * from q2Results;
```

▸ (2) Spark Jobs

	TrueFalse	maxAmount
1	true	9998.2
2	false	9999.55

Showing all 2 rows.

Command took 1.95 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 12:33:17 AM on My Cluster

Cmd 8

Question 3: Avg Function

Summary

Compute the average of the `Amount` field for each unique value in the `TrueFalse` field in the table `revenue3`.

Steps to complete

- Computes the average of `Amount` for `True` records and `False` records from the `TrueFalse` field in the table `revenue3`.
- Renames the new column to `avgAmount`
- Store the records in a temporary view named `q3Results` with the following schema:

column	type
TrueFalse	boolean
avgAmount	double

A properly completed solution should produce a DataFrame similar to this sample output:

TrueFalse	AvgAmount
true	2243937.93
false	2559457.1799999997

Cmd 9

```
1  -- TODO  Answer 3
2  Create or Replace Temporary View q3Results as (
3    Select TrueFalse, avg(Amount) as AvgAmount
4    From revenue3
5    Group by TrueFalse
6  );
7  Select * from q3Results;
```

▸ (4) Spark Jobs

	TrueFalse	AvgAmount
1	true	5057.301391181796
2	false	5049.06687264057

Showing all 2 rows.

Command took 3.71 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 12:34:55 AM on My Cluster

Question 4: Pivot

Summary

Calculate the total `Amount` for `YesNo` values of **true** and **false** in 2002 and 2003 from the table `revenue4`.

Steps to complete

- Casts the `UTCTime` field to Timestamp and names the new column `Date`
- Extracts a `Year` column from the `Date` column
- Filters for years greater than 2001 and less than or equal to 2003
- Groups by `YesNo` and creates a pivot table to get the total `Amount` for each year and each value in `YesNo`
- Represents each total amount as a float rounded to two decimal places
- Store the results into a temporary table named `q4results`

A properly completed solution should produce a view similar to this sample output:

YesNo	2002	2003
true	61632.3	8108.47
false	44699.99	35062.22

```sql
-- TODO  Answer 4
CREATE OR REPLACE TEMPORARY VIEW q4Results AS
  SELECT *
  FROM (SELECT Year, YesNo, Amount
        FROM (SELECT year(CAST(UTCTime AS timestamp)) as Year,
                     YesNo,
                     Amount
              FROM revenue4)
        WHERE Year IN (2002, 2003)
        )
  PIVOT ( round( sum(Amount), 2) AS total FOR Year in (2002, 2003) );

  SELECT * FROM q4Results;
```

▸ (5) Spark Jobs

	YesNo	2002	2003
1	true	1289367.34	1119580.12
2	false	1190882.92	1195662.22

Showing all 2 rows.

Command took 4.05 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 11:43:01 PM on My Cluster

Question 5: Null Values and Aggregates

Summary

Compute sums of `amount` grouped by `aisle` after dropping null values from `products` table.

Steps to complete

- Drops any rows that contain null values in either the `itemId` or the `aisle` column
- Aggregates sums of the `amount` column grouped by `aisle`
- Store the results into a temporary view named `q5Results`

Cmd 13

```
1  -- TODO  Answer 5
2  Create or Replace Temporary View q5Results as (
3    Select aisle, sum(amount) TotalAmount
4    From products
5    where itemId is not null and aisle is not null
6    group by aisle
7  );
8  select * from q5Results;
```

▸ (2) Spark Jobs

	aisle	TotalAmount
1	3	63
2	5	14
3	7	107
4	12	56
5	2	126
6	8	8

Showing all 6 rows.

Command took 0.97 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 12:43:49 AM on My Cluster

Question 6: Generate Subtotals By Rollup

Summary

Compute averages of `income` grouped by `itemName` and `month` such that the results include averages across all months as well as a subtotal for an individual month from the `sales` table.

Steps to complete

- Coalesces null values in the `month` column generated by the `ROLLUP` clause
- Store the results into a temporary view named `q6Results`

Your results should look something like this:

itemName	month	avgRevenue
Anim	10	4794.16
Anim	7	5551.31
Anim	All months	5046.54
Aute	4	4069.51
Aute	7	3479.31
Aute	8	6339.28
Aute	All months	4489.41
...	...	...

```
1  --TODO  Answer 6
2  Create or Replace Temporary View q6Results as (
3  Select
4    COALESCE(itemName, "All items") as itemName,
5    COALESCE(month(date), "All months") as month,
6    ROUND(AVG(Revenue), 2) AvgRevenue
7  from
8    sales
9  group by rollup (itemName, month(date))
10 order by itemName, month
11 );
12 --select itemName, coalesce(month, "All months") as month, AvgRevenue from q6Results;
13 select * from q6Results;
```

▸ (4) Spark Jobs

	itemName	month	AvgRevenue
1	Ad	10	848.8
2	Ad	2	7968.57
3	Ad	3	3497.96
4	Ad	6	3105.51
5	Ad	All months	3783.76
6	Adipiscing	12	9787.74
7	Adipiscing	3	6744.76

Truncated results, showing first 1000 rows.

Command took 3.50 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 11:44:51 PM on My Cluster

06 - Nested Data (SQL)

Cmd 2

Create Tables

Run the cell below to create tables for the questions in this notebook.

Cmd 3

```
1  %run ../Utilities/06-CreateTables
```

```
Command took 8.20 seconds -- by SWCPROPERTY@GMAIL.COM at 7/9/2021, 11:45:56 PM on My Cluster
```

Declared the following table:

- **databricksBlog**

Cmd 4

Question 1: Array and Explode

Summary

Get a distinct list of authors who have contributed blog posts in the "Company Blog" category.

Steps to complete

Write a SQL query that achieves the following:

- Explodes the `authors` field to create an `author` field that contains only one author per row in table `databricksBlog`.
- Limits records to contain **only** unique authors
- Filters records where `categories` include "Company Blog"
- Store the results in a temporary view named `results`

A properly completed solution should produce a view named `results` similar to this sample output:

author	categories
Anthony Joseph	["Announcements", "Company Blog"]
Vida Ha	["Company Blog", "Product"]
Nan Zhu (Chief Architect at Faimdata)	["Company Blog", "Product"]

Cmd 5

```
1  --TODO  Answer 1
2  CREATE OR REPLACE TEMPORARY VIEW results AS
3    SELECT
4     DISTINCT (EXPLODE(authors)) AS author,
5     categories
6    FROM databricksBlog
7    WHERE array_contains(categories, "Company Blog");
8
9  SELECT * FROM results;
10
```

▸ (2) Spark Jobs

	author	categories
1	Anthony Joseph	▸ ["Announcements", "Company Blog"]
2	Vida Ha	▸ ["Company Blog", "Product"]
3	Nan Zhu (Chief Architect at Faimdata)	▸ ["Company Blog", "Partners"]
4	Russell Cardullo (Sharethrough)	▸ ["Company Blog", "Customers"]
5	Eric Carr (VP Core Systems Group at Guavus)	▸ ["Company Blog", "Partners"]
6	Paco Nathan	▸ ["Announcements", "Company Blog"]
7	Sonal Goyal (CEO)	▸ ["Company Blog", "Partners"]

Showing all 58 rows.

Command took 0.56 seconds -- by SWCPROPERTY@GMAIL.COM at 7/10/2021, 12:06:38 AM on My Cluster

Cmd 6

© 2020 Databricks, Inc. All rights reserved.
Apache, Apache Spark, Spark and the Spark logo are trademarks of the Apache Software Foundation.

07 - Higher Order Functions (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear

Cmd 2

Create Tables

Run the cell below to create tables for the questions in this notebook.

Cmd 3

```
1  %run ../Utilities/07-CreateTables
```

```
Command took 6.39 seconds -- by SWCPROPERTY@GMAIL.COM at 7/10/2021, 12:08:25 AM on My Cluster
```

▸ dataDF: pyspark.sql.dataframe.DataFrame = [id: long, firstName: string ... 2 more fields]

Declared the following table:
- **finances**

Declared the following table:
- **charges**

Cmd 4

Question 1: Transform

Summary

Use the `TRANSFORM` function and the table `finances` to calculate `interest` for all cards issued to each user.

Steps to complete

Write a SQL query that achieves the following:
- Displays cardholder's `firstName`, `lastName`, and a new column named `interest`
- Uses `TRANSFORM` to extract charges for each card in the expenses column and calculates interest owed assuming a rate of 6.25%.
- Stores the new values as an array in the interest column
- Stores results in a temporary table named `q1Results`

A properly completed solution should return a view that looks similar to this:

firstName	lastName	interest
Lance	Da Costa	[138.9, 373.55, 158.97]

```
1  --TODO  Answer 1
2  select * from finances
```

▸ (3) Spark Jobs

	id	firstName	lastName	expenses
1	1	Lance	Da Costa	▸ [{"lastPayment": "2020-07-18", "paymentDue": "2020-07-18", "charges": "2222.46", "currency": "USD", "cardType": "jcb"}, {"lastPayment": "2020-07-01", "paymentDue": "2020-07-22", "charges": "5976.76", "currency": "USD", "cardType": "jcb"}, {"lastPayment": "2020-07-12", "paymentDue": "2020-07-22", "charges": "2543.55", "currency": "USD", "cardType": "visa"}]
2	2	Emilie	Newlove	▸ [{"lastPayment": "2020-07-10", "paymentDue": "2020-07-08", "charges": "6344.33", "currency": "USD", "cardType": "bankcard"}]
3	3	Alvy	Records	▸ [{"lastPayment": "2020-07-02", "paymentDue": "2020-07-29", "charges": "9170.83", "currency": "USD", "cardType": "mastercard"}, {"lastPayment": "2020-07-06", "paymentDue": "2020-07-22", "charges": "3201.67", "currency": "USD", "cardType": "maestro"}, {"lastPayment": "2020-07-28", "paymentDue": "2020-07-08", "charges": "6087.61", "currency": "USD", "cardType": "americanexpress"}, {"lastPayment": "2020-07-12", "paymentDue": "2020-07-21", "charges": "3392.61", "currency": "USD", "cardType": "americanexpress"}]
4	4	Jena	Fairley	▸ [{"lastPayment": "2020-07-25", "paymentDue": "2020-07-17", "charges": "9726.16", "currency": "USD", "cardType": "jcb"}, {"lastPayment": "2020-07-27", "paymentDue": "2020-07-14", "charges": "2578.18", "currency": "USD", "cardType": "americanexpress"}]
5	5	Klarika	Pady	▸ [{"lastPayment": "2020-07-12", "paymentDue": "2020-07-17", "charges": "8460.46", "currency": "USD", "cardType": "bankcard"}, {"lastPayment": "2020-07-11", "paymentDue": "2020-07-08", "charges": "8573.72", "currency": "USD", "cardType": "bankcard"}]

Showing all 5 rows.

Command took 0.89 seconds -- by SWCPROPERTY@GMAIL.COM at 7/10/2021, 12:16:56 AM on My Cluster

```
1  --TO DO ANSWER
2  SELECT firstName, lastName,
3  TRANSFORM(expenses, card -> ROUND(card.charges * 0.0625, 2)) AS interest
4  FROM finances
5
```

▸ (3) Spark Jobs

	firstName	lastName	interest
1	Lance	Da Costa	▸ [138.9, 373.55, 158.97]
2	Emilie	Newlove	▸ [396.52]
3	Alvy	Records	▸ [573.18, 200.1, 380.48, 212.04]
4	Jena	Fairley	▸ [607.89, 161.14]
5	Klarika	Pady	▸ [528.78, 535.86]

Showing all 5 rows.

Command took 1.06 seconds -- by SWCPROPERTY@GMAIL.COM at 7/10/2021, 12:16:14 AM on My Cluster

Question 2: Exists

Summary

Use the table from Question 1, `finances`, to flag users whose records indicate that they made a late payment.

Steps to complete

Write a SQL query that achieves the following:

- Displays cardholder's `firstName`, `lastName`, and a new column named `lateFee`
- Uses the EXISTS function to flag customers who have made been charged a late payment fee.
- Store the results in a temporary view named `q2Results`

A properly completed solution should return a DataFrame that looks similar to this:

firstName	lastName	lateFee
Lance	DaCosta	true

Cmd 7

```
--TO DO ANSWER
CREATE OR REPLACE TEMPORARY VIEW q2Results AS
  SELECT firstName, lastName,
  EXISTS(expenses, card -> TO_DATE(card.paymentDue) > TO_DATE(card.lastPayment)) AS lateFee
  FROM finances;

SELECT * FROM q2Results
```

▸ (3) Spark Jobs

	firstName	lastName	lateFee
1	Lance	Da Costa	true
2	Emilie	Newlove	false
3	Alvy	Records	true
4	Jena	Fairley	false
5	Klarika	Pady	true

Showing all 5 rows.

Command took 0.46 seconds -- by SWCPROPERTY@GMAIL.COM at 7/10/2021, 12:22:03 AM on My Cluster

Question 3: Reduce

Summary

Use the `REDUCE` function to produce a query on the table `charges` that calculates total charges in dollars and total charges in Japanese Yen.

Steps to complete

Write a SQL query that achieves the following:

- Uses the `REDUCE` function to calculate the total charges in US Dollars (given)
- Uses the `REDUCE` function to convert the total charges to Japanese Yen using a conversion rate where 1 USD = 107.26 JPY
- Stores the results in a temporary table named `q3Results`

NOTE: In the `REDUCE` function, the accumulator must be of the same type as the input. You will have to `CAST` the accumulator as a `DOUBLE` to use this function with this data. example: `CAST (0 AS DOUBLE)`

A properly completed solution should return a DataFrame that looks similar to this:

firstName	lastName	allCharges	totalDollars	totalYen
Lance	DaCosta	["2222.46", "5976.76", "2543.55"]	10742.77	1152269.51

Cmd 9

```
--TO DO ANSWER
CREATE OR REPLACE TEMPORARY VIEW q3Results AS
  SELECT firstName, lastName, allCharges,
  REDUCE (allCharges, CAST(0 AS DOUBLE), (charge, acc) -> charge + acc) AS totalDollars,
  REDUCE (allCharges, CAST(0 AS DOUBLE), (charge, acc) -> charge + acc, acc -> ROUND(acc * 107.26, 2)) AS totalYen
  FROM charges;

SELECT * FROM q3Results

```

▸ (3) Spark Jobs

	firstName	lastName	allCharges	totalDollars	totalYen
1	Lance	Da Costa	▸ ["2222.46", "5976.76", "2543.55"]	10742.77	1152269.51
2	Emilie	Newlove	▸ ["6344.33"]	6344.33	680492.84
3	Alvy	Records	▸ ["9170.83", "3201.67", "6087.61", "3392.61"]	21852.72	2343922.75
4	Jena	Fairley	▸ ["9726.16", "2578.18"]	12304.34	1319763.51
5	Klarika	Pady	▸ ["8460.46", "8573.72"]	17034.18	1827086.15