databricks6.3 Partitioning Tables

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.

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

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
```

OK

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

```
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 DeviceData;

SELECT * FROM AvgTemps;
```

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

Truncated results, showing first 1000 rows.

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.

```
SHOW PARTITIONS AvgTemps
```

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

Showing all 4 rows.

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 (https://docs.databricks.com/notebooks/widgets.html).

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.

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

OK

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.

```
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
```

	device_type	avgTemp	stdTemp
1	sensor-inest	15.4804	4.25

Showing all 1 rows.

Remove widget

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

```
REMOVE WIDGET selectedDeviceType
```

OK

```
CREATE WIDGET DROPDOWN cuts DEFAULT "Good" CHOICES
SELECT DISTINCT cut
FROM diamonds
```

```
SELECT COUNT(*) AS numChoices, getArgument("cuts") AS cuts
FROM diamonds WHERE cut = getArgument("cuts")
```

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.

```
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";
```

	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
6	dc-102	8	23	16.870967741935484
7	dc-102	8	18	16.870967741935484

Showing all 124 rows.

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.

```
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;
```

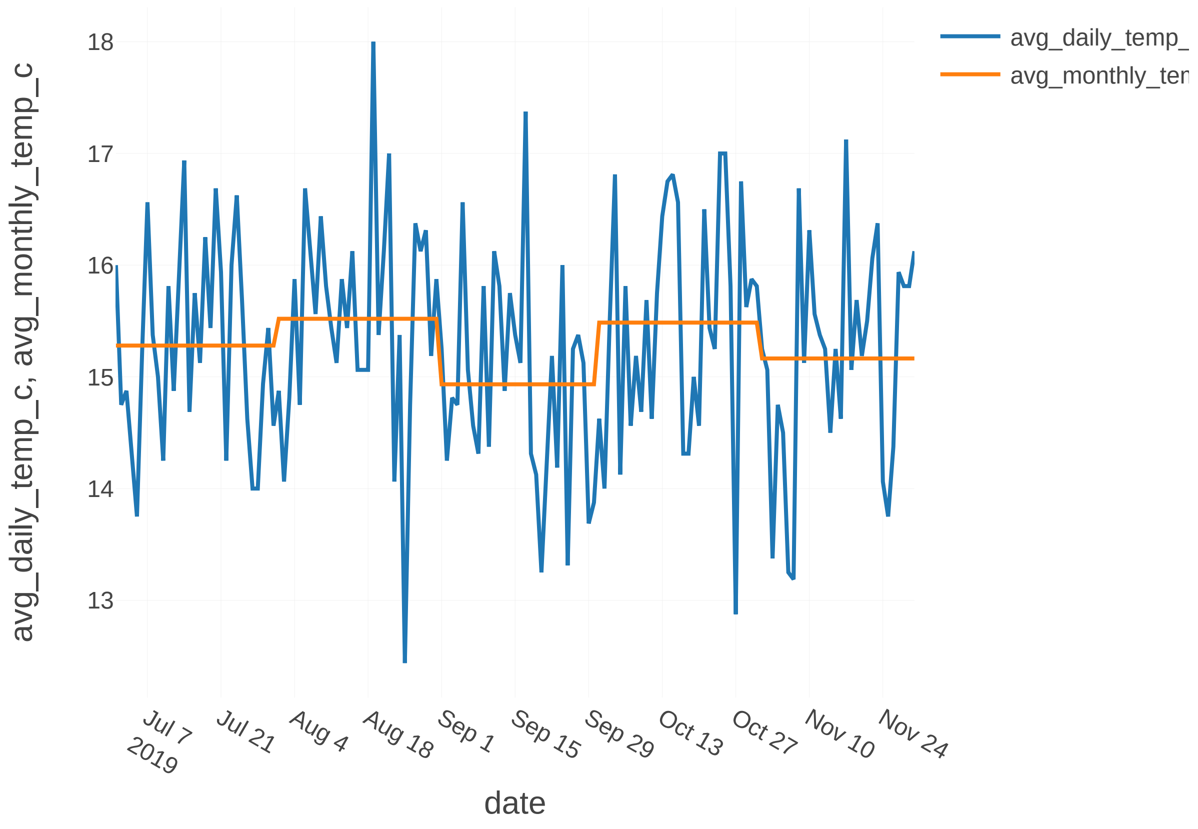
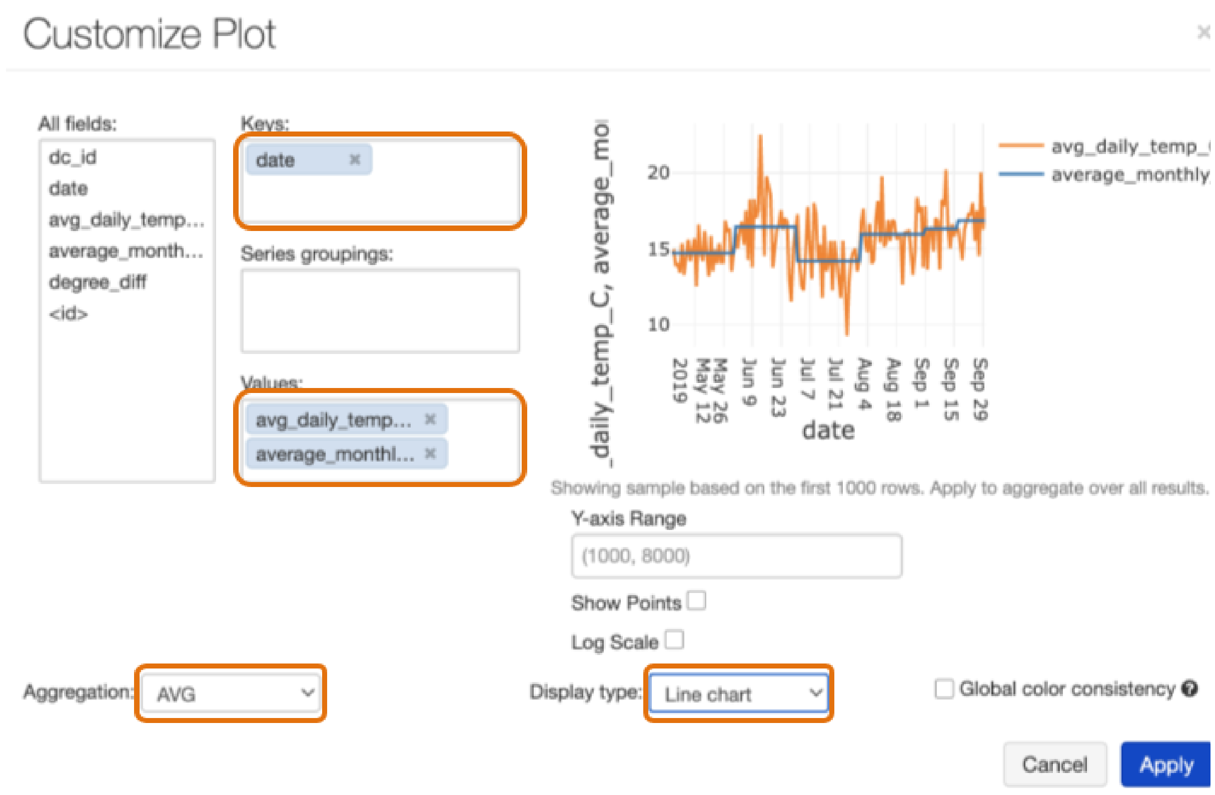


Chart results

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


Customize Plot

All fields: dc_id, date, avg_daily_temp..., average_month..., degree_diff, <id>

Keys: date

Series groupings:

Values: avg_daily_temp..., average_monthl...

Showing sample based on the first 1000 rows. Apply to aggregate over all results.

Y-axis Range: (1000, 8000)

Show Points ☐

Log Scale ☐

Aggregation: AVG

Display type: Line chart

☐ Global color consistency

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`.

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