databricks3.2 Basic Queries

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.

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.

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

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 (https://docs.databricks.com/data/tables.html) 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 (https://databricks.com/glossary/what-is-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.

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

OK

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

```
SELECT * FROM People10M;
```

	id	firstName	middleName	lastName	gender	birth

1	1	Pennie	Carry	Hirschmann	F	1955
2	2	An	Amira	Cowper	F	1992
3	3	Quyen	Marlen	Dome	F	1970
4	4	Coralie	Antonina	Marshal	F	1990
5	5	Terrie	Wava	Bonar	F	1980
6	6	Chassidy	Concepcion	Bourthouloume	F	1990
7	7	Geri	Tambra	Mosby	F	1970

Truncated results, showing first 1000 rows.

```
select *, row_number from People10M where row_number = 4;
```

```
Error in SQL statement: AnalysisException: cannot resolve '`row_number`' giv
en input columns: [spark_catalog.default.people10m.birthDate, spark_catalog.
default.people10m.firstName, spark_catalog.default.people10m.gender, spark_c
atalog.default.people10m.id, spark_catalog.default.people10m.lastName, spark
_catalog.default.people10m.middleName, spark_catalog.default.people10m.salar
y, spark_catalog.default.people10m.ssn]; line 1 pos 42;
'Project [*, 'row_number]
+- 'Filter ('row_number = 4)
   +- SubqueryAlias spark_catalog.default.people10m
      +- Relation[id#240,firstName#241,middleName#242,lastName#243,gender#24
4,birthDate#245,ssn#246,salary#247] parquet
```

```
Error in SQL statement: ParseException:
missing 'FUNCTIONS' at '<EOF>'(line 1, pos 8)

== SQL ==
show row
--------^^^
```

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

```
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.

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`.

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

	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
8	Chung	Dian	Dautry	1998-01-12T05:00:00.000+0000
9	Erica	Louvenia	O'Drought	1991-03-08T05:00:00.000+0000

10	Katelyn	Merrie	Pocklington	1994-01-16T05:00:00.000+0000
11	Monserrate	Goldie	Trimming	1993-09-17T04:00:00.000+0000
12	Deanne	Lindsay	Lambdin	1999-07-28T04:00:00.000+0000
13	Carey	Kareen	Ciric	1999-02-23T05:00:00.000+0000
14	In	Susann	Lacroix	1999-07-23T04:00:00.000+0000

Math

Spark SQL includes many built-in functions (https://spark.apache.org/docs/latest/api/sql/) 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`.

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

	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.

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

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.

```
SELECT * FROM PeopleSavings;
```

	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.

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.

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

	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.

Define a new table

Now we will show you how to create a table using Parquet. Parquet (https://databricks.com/glossary/what-is-parquet#:~:text=Parquet%20is%20an%20open%20source,like%20CSV%20or%20TSV

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.

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

OK

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.

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

	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.

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.

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.

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

	count(DISTINCT firstName)
1	93889

Showing all 1 rows.

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

	count(DISTINCT firstName)
1	5113

Showing all 1 rows.

Create temporary views

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

```
CREATE OR REPLACE TEMPORARY VIEW SSADistinctNames AS
  SELECT DISTINCT firstName AS ssaFirstName
  FROM SSANames;

CREATE OR REPLACE TEMPORARY VIEW PeopleDistinctNames AS
  SELECT DISTINCT firstName
  FROM People10M
```

OK

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 (https://en.wikipedia.org/wiki/Join_(SQL) 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.

```
SELECT firstName
FROM PeopleDistinctNames
JOIN SSADistinctNames ON firstName = ssaFirstName
```

	firstName
1	Susanna
2	Julianne
3	Lashanda

4	Kiana
5	Tyler
6	Sandi
7	Faye

Truncated results, showing first 1000 rows.

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.

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

	count(1)
1	5096

Showing all 1 rows.

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